RFCs in Hypertext

Volume 1 Number 1
1 October 1991

Choose a topic below to access material giving a general overview of the Internet, information relating to a particular protocol layer, or a specific RFC. Please refer to the Disclaimer for important information before using this system.

Introduction

This system is intended to provide guidance for vendors, implementors, and users of Internet communication software. The inlcuded material represents the consensus of a large body of technical experience and wisdom, contributed by the members of the Internet research and vendor communities.

This system enumerates standard protocols that a host connected to the Internet must use, and it includes the Request For Comments (RFCs) and other documents describing the current specifications for these protocols. It corrects errors in the included documents and adds additional discussion and guidance for an implementor. These corrections and additions have been incorporated into the original text of the relevant RFCs both directly and via links to other documents.

The specifications of this system must be followed to meet the general goal of arbitrary host interoperation across the diversity and complexity of the Internet system. Although most current implementations fail to meet these requirements in various ways, some minor and some major, this specification is the ideal towards which we need to move.

These requirements are based on the current level of Internet architecture. This system will be updated on a regular basis to provide additional clarifications and to include new and/or additional information in those areas in which specifications are still evolving.

Overview

Link Layer

Internet Layer

Transport Layer

TCP Specific Issues

Application layer

User Level Protocols

- **Finger** (User Information)
- **FTP** (File Transfer Protocol)
- **NICName/Whois**
- **POP-2** (Post Office Protocol - Version 2)
- **POP-3** (Post Office Protocol - Version 3)
- **SMTP** (Electronic Mail Delivery)
- **Telnet** (Remote Login)
- **TFTP** (Trivial File Transfer Protocol)

Support Protocols

- **Cooperative Processing**
  - **XDR** (External Data Representation)
  - **RPC** (Remote Procedure Calls)
  - **NFS** (Network File System)
- **Host Initialization**
  - **BOOTP** (Bootstrap Protocol)
  - **ARP** (Address Resolution Protocol)
  - **RARP** (Reverse Address Resolution Protocol)
- **Domain Name System**
- **Network Management**
  - **Structure and ID of Management Information Base**
  - **Management Information Base (MIB I)**
  - **Management Information Base (MIB II)**
  - **Concise MIB Definitions**
  - **SNMP** (Simple Network Management Protocol)
    - **Protocol Walk-Through**

Security Considerations

There are many security issues in the communication layers of host software, but a full discussion is beyond the scope of this system.

The Internet architecture generally provides little protection against spoofing of IP source addresses, so any security mechanism that is based upon verifying the IP source address of a datagram should be treated with suspicion. However, in restricted environments some source-address checking may be possible. For example, there might be a secure LAN whose gateway to the rest of the Internet discarded any incoming datagram with a source address that spoofed the LAN address. In this case, a host on the LAN could use the source address to test for local vs. remote source. This problem is complicated by source routing, and some have suggested that source-routed datagram forwarding by hosts should be outlawed for security reasons.

Security-related issues are mentioned in sections concerning the IP Security option, the ICMP Parameter Problem Message, IP options in UDP datagrams, and reserved TCP ports.

It is standard practice in RFC authoring to include a section titled "Security Considerations", although in most cases the text of this section reads "Security issues are not addressed in this memo.". These sections have been eliminated from the included documents.

Disclaimer

Neither InfoMagic, nor the Internet Engineering Task Force, nor the Internet Activities Board, nor the United States Government, nor the National Science Foundation, nor any of their employees makes any warranty or assumes the legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. References to any special commercial products, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the above mentioned parties. The views and opinions of the author(s) do not necessarily state or reflect those of the parties named above and shall not be used for advertising or product endorsement.

InfoMagic has proceeded in good faith to accurately present both the original protocol specifications and all of the updates and corrections to those specifications available at the time of publication. We cannot, however, warranty that we have succeeded in this effort. Errors and omissions will be corrected in future releases of this system as we become aware of them.

Overview

The Internet Architecture

General background and discussion on the Internet architecture and supporting protocol suite can be found in the DDN Protocol Handbook [INTRO:3]; for background see for example [INTRO:9], [INTRO:10], and [INTRO:11]. Reference [INTRO:5] describes the procedure for obtaining Internet protocol documents, while RFC-1060 contains a list of the numbers assigned within Internet protocols.

Overview

Internet Hosts

A host computer, or simply "host," is the ultimate consumer of communication services. A host generally executes application programs on behalf of user(s), employing network and/or Internet communication services in support of this function. An Internet host corresponds to the concept of an "End-System" used in the OSI protocol suite [INTRO:13].

An Internet communication system consists of interconnected packet networks supporting communication among host computers using the Internet protocols. The networks are interconnected using packet-switching computers called "gateways" or "IP routers" by the Internet community, and "Intermediate Systems" by the OSI world [INTRO:13]. The RFC "Requirements for Internet Gateways" [RFC-1009] contains the official specifications for Internet gateways.

Internet hosts span a wide range of size, speed, and function. They range in size from small microprocessors through workstations to mainframes and supercomputers. In function, they range from single-purpose hosts (such as terminal servers) to full-service hosts that support a variety of online network services, typically including remote login, file transfer, and electronic mail.

Overview

Architectural Assumptions

The current Internet architecture is based on a set of assumptions about the communication system. The assumptions most relevant to hosts are as follows:

(a) The Internet is a network of networks.

Each host is directly connected to some particular network(s); its connection to the Internet is only conceptual. Two hosts on the same network communicate with each other using the same set of protocols that they would use to communicate with hosts on distant networks.

(b) Gateways don't keep connection state information.

To improve robustness of the communication system, gateways are designed to be stateless, forwarding each IP datagram independently of other datagrams. As a result, redundant paths can be exploited to provide robust service in spite of failures of intervening gateways and networks.
All state information required for end-to-end flow control and reliability is implemented in the hosts, in the transport layer or in application programs. All connection control information is thus co-located with the end points of the communication, so it will be lost only if an end point fails.

(c) Routing complexity should be in the gateways.

Routing is a complex and difficult problem, and ought to be performed by the gateways, not the hosts. An important objective is to insulate host software from changes caused by the inevitable evolution of the Internet routing architecture.

(d) The System must tolerate wide network variation.

A basic objective of the Internet design is to tolerate a wide range of network characteristics -- e.g., bandwidth, delay, packet loss, packet reordering, and maximum packet size. Another objective is robustness against failure of individual networks, gateways, and hosts, using whatever bandwidth is still available. Finally, the goal is full "open system interconnection": an Internet host must be able to interoperate robustly and effectively with any other Internet host, across diverse Internet paths.

Sometimes host implementors have designed for less ambitious goals. For example, the LAN environment is typically much more benign than the Internet as a whole; LANs have low packet loss and delay and do not reorder packets. Some vendors have fielded host implementations that are adequate for a simple LAN environment, but work badly for general interoperation. The vendor justifies such a product as being economical within the restricted LAN market. However, isolated LANs seldom stay isolated for long; they are soon gatewayed to each other, to organization-wide internets, and eventually to the global Internet system. In the end, neither the customer nor the vendor is served by incomplete or substandard Internet host software.

The requirements spelled out in this document are designed for a full-function Internet host, capable of full interoperation over an arbitrary Internet path.

Overview

Internet Protocol Suite

To communicate using the Internet system, a host must implement the layered set of protocols comprising the Internet protocol suite. A host typically must implement at least one protocol from each layer.

The protocol layers used in the Internet architecture are as follows [RFC-1011]:

Application Layer
Transport Layer
Internet Layer
Link Layer

Overview

Application Layer Protocols

The application layer is the top layer of the Internet protocol suite. The Internet suite does not further subdivide the application layer, although some of the Internet application layer protocols do contain some internal sub-layering. The application layer of the Internet suite essentially combines the functions of the top two layers -- Presentation and Application -- of the OSI reference model.

We distinguish two categories of application layer protocols: user protocols that provide service directly to users, and support protocols that provide common system functions.

The most common Internet user protocols are:

- **Telnet** (remote login)
- **FTP** (file transfer)
- **TFTP** (trivial file transfer)
- **SMTP** (electronic mail delivery)

There are a number of other standardized user protocols and many private user protocols. Refer to "IAB Official Protocols" [RFC-1250] for the current status of these protocols.

Support protocols, used for host name mapping, booting, and management, include SNMP, BOOTP, RARP, and the Domain Name System (DNS) protocols. See Application Layer General Issues for an overview.

Overview

Transport Layer Protocols

The transport layer provides end-to-end communication services for applications. There are two primary transport layer protocols at present:

- Transmission Control Protocol (TCP)
- User Datagram Protocol (UDP)

TCP is a reliable connection-oriented transport service that provides end-to-end reliability, resequencing, and flow control. UDP is a connectionless ("datagram") transport service.

Other transport protocols have been developed by the research community, and the set of official Internet transport protocols may be expanded in the future.

Overview

Internet Layer Protocols

All Internet transport protocols use the Internet Protocol (IP) to carry data from source host to destination host. IP is a connectionless or datagram internetwork service, providing no end-to-end delivery guarantees. Thus, IP datagrams may arrive at the destination host damaged, duplicated, out of order, or not at all. The layers above IP are responsible for reliable delivery service when it is required. The IP protocol includes provision for addressing, type-of-service specification, fragmentation and reassembly, and security information.

The datagram or connectionless nature of the IP protocol is a fundamental and characteristic feature of the Internet architecture. Internet IP was the model for the OSI Connectionless Network Protocol [INTRO:12].

ICMP is a control protocol that is considered to be an integral part of IP, although it is architecturally layered upon IP, i.e., it uses IP to carry its data end- to-end just as a transport protocol like TCP or UDP does. ICMP provides error reporting, congestion reporting, and first-hop gateway redirection.

IGMP is an Internet layer protocol used for establishing dynamic host groups for IP multicasting.

Overview

Embedded Gateway Code

Some Internet host software includes embedded gateway functionality, so that these hosts can forward packets as a gateway would, while still performing the application layer functions of a host.

Such dual-purpose systems must follow the Gateway Requirements RFC [RFC-1009] with respect to their gateway functions, and must follow the present document with respect to their host functions. In all overlapping cases, the two specifications should be in agreement.

There are varying opinions in the Internet community about embedded gateway functionality. The main arguments are as follows:

Pro: In a local network environment where networking is informal, or in isolated internets, it may be convenient and economical to use existing host systems as gateways.

There is also an architectural argument for embedded gateway functionality: multihoming is much more common than originally foreseen, and multihoming forces a host to make routing decisions as if it were a gateway. If the multihomed host contains an embedded gateway, it will have full routing knowledge and as a result will be able to make more optimal routing decisions.

Con: Gateway algorithms and protocols are still changing, and they will continue to change as the Internet system grows larger. Attempting to include a general gateway function within the host IP layer will force host system maintainers to track these (more frequent) changes. Also, a larger pool of gateway implementations will make coordinating the changes more difficult. Finally, the complexity of a gateway IP layer is somewhat greater than that of a host, making the implementation and operation tasks more complex.

In addition, the style of operation of some hosts is not appropriate for providing stable and robust gateway service.

There is considerable merit in both of these viewpoints. One conclusion can be drawn: an host administrator must have conscious control over whether or not a given host acts as a gateway. See Internet Layer Protocols for the detailed requirements.

General Considerations

There are two important lessons that vendors of Internet host software have learned and which a new vendor should consider seriously.

Overview

Continuing Internet Evolution

The enormous growth of the Internet has revealed problems of management and scaling in a large datagram-based packet communication system. These problems are being addressed, and as a result there will be continuing evolution of the specifications described in this document. These changes will be carefully planned and controlled, since there is extensive participation in this planning by the vendors and by the organizations responsible for operations of the networks.

Development, evolution, and revision are characteristic of computer network protocols today, and this situation will persist for some years. A vendor who develops computer communication software for the Internet protocol suite (or any other protocol suite!) and then fails to maintain and update that software for changing specifications is going to leave a trail of unhappy customers. The Internet is a large communication network, and the users are in constant contact through it. Experience has shown that knowledge of deficiencies in vendor software propagates quickly through the Internet technical community.

Overview

Robustness Principle

At every layer of the protocols, there is a general rule whose application can lead to enormous benefits in robustness and interoperability. The following principle originally appeared in [RFC-791]:

"Be liberal in what you accept, and conservative in what you send"

Software should be written to deal with every conceivable error, no matter how unlikely; sooner or later a packet will come in with that particular combination of errors and attributes, and unless the software is prepared, chaos can ensue. In general, it is best to assume that the network is filled with malevolent entities that will send in packets designed to have the worst possible effect. This assumption will lead to suitable protective design, although the most serious problems in the Internet have been caused by unenvisaged mechanisms triggered by low-probability events; mere human malice would never have taken so devious a course!

Adaptability to change must be designed into all levels of Internet host software. As a simple example, consider a protocol specification that contains an enumeration of values for a particular header field -- e.g., a type field, a port number, or an error code; this enumeration must be assumed to be incomplete. Thus, if a protocol specification defines four possible error codes, the software must not break when a fifth code shows up. An undefined code might be logged, but it must not cause a failure.

The second part of the principle is almost as important: software on other hosts may contain deficiencies that make it unwise to exploit legal but obscure protocol features. It is unwise to stray far from the obvious and simple, lest untoward effects result elsewhere. A corollary of this is "watch out for misbehaving hosts"; host software should be prepared, not just to survive other misbehaving hosts, but also to cooperate to limit the amount of disruption such hosts can cause to the shared communication facility.

Overview

Error Logging

The Internet includes a great variety of host and gateway systems, each implementing many protocols and protocol layers, and some of these contain bugs and mis-features in their Internet protocol software. As a result of complexity, diversity, and distribution of function, the diagnosis of Internet problems is often very difficult.

Problem diagnosis will be aided if host implementations include a carefully designed facility for logging erroneous or "strange" protocol events. It is important to include as much diagnostic information as possible when an error is logged. In particular, it is often useful to record the header(s) of a packet that caused an error. However, care must be taken to ensure that error logging does not consume prohibitive amounts of resources or otherwise interfere with the operation of the host.

There is a tendency for abnormal but harmless protocol events to overflow error logging files; this can be avoided by using a "circular" log, or by enabling logging only while diagnosing a known failure. It may be useful to filter and count duplicate successive messages. One strategy that seems to work well is: (1) always count abnormalities and make such counts accessible through the management protocol; and (2) allow the logging of a great variety of events to be selectively enabled. For example, it might useful to be able to "log everything" or to "log everything for host X".

Note that different managements may have differing policies about the amount of error logging that they want normally enabled in a host. Some will say, "if it doesn't hurt me, I don't want to know about it", while others will want to take a more watchful and aggressive attitude about detecting and removing protocol abnormalities.

Overview

Configuration

It would be ideal if a host implementation of the Internet protocol suite could be entirely self-configuring. This would allow the whole suite to be implemented in ROM or cast into silicon, it would simplify diskless workstations, and it would be an immense boon to harried LAN administrators as well as system vendors. We have not reached this ideal; in fact, we are not even close.

At many points in this document, you will find a requirement that a parameter be a configurable option. There are several different reasons behind such requirements. In a few cases, there is current uncertainty or disagreement about the best value, and it may be necessary to update the recommended value in the future. In other cases, the value really depends on external factors -- e.g., the size of the host and the distribution of its communication load, or the speeds and topology of nearby networks -- and self-tuning algorithms are unavailable and may be insufficient. In some cases, configurability is needed because of administrative requirements.

Finally, some configuration options are required to communicate with obsolete or incorrect implementations of the protocols, distributed without sources, that unfortunately persist in many parts of the Internet. To make correct systems coexist with these faulty systems, administrators often have to "mis- configure" the correct systems. This problem will correct itself gradually as the faulty systems are retired, but it cannot be ignored by vendors.

When we say that a parameter must be configurable, we do not intend to require that its value be explicitly read from a configuration file at every boot time. We recommend that implementors set up a default for each parameter, so a configuration file is only necessary to override those defaults that are inappropriate in a particular installation. Thus, the configurability requirement is an assurance that it will be **possible** to override the default when necessary, even in a binary-only or ROM-based product.

This document requires a particular value for such defaults in some cases. The choice of default is a sensitive issue when the configuration item controls the accommodation to existing faulty systems. If the Internet is to converge successfully to complete interoperability, the default values built into implementations must implement the official protocol, not "mis-configurations" to accommodate faulty implementations. Although marketing considerations have led some vendors to choose mis-configuration defaults, we urge vendors to choose defaults that will conform to the standard.

Finally, we note that a vendor needs to provide adequate documentation on all configuration parameters, their limits and effects.

Overview

Requirements

In this document, the words that are used to define the significance of each particular requirement appear in **bold face**.

These words are:

* **"must"**

This word or the adjective **"required"** means that the item is an absolute requirement of the specification.

* **"should"**

This word or the adjective **"recommended"** means that there may exist valid reasons in particular circumstances to ignore this item, but the full implications should be understood and the case carefully weighed before choosing a different course.

* **"may"**

This word or the adjective **"optional"** means that this item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because it enhances the product, for example; another vendor may omit the same item.

An implementation is not compliant if it fails to satisfy one or more of the **must** requirements for the protocols it implements. An implementation that satisfies all the **must** and all the **should** requirements for its protocols is said to be "unconditionally compliant"; one that satisfies all the **must** requirements but not all the **should** requirements for its protocols is said to be "conditionally compliant".

Overview

Terminology

This document uses the following technical terms:

Segment
Message
IP Datagram
Packet
Frame
Connected Network
Multihomed
Physical network interface
Logical [network] interface
Specific-destination address
Path
MTU

Acknowledgments

RFC-1122 and RFC-1123 which formed the basis for this system incorporate contributions and comments from a large group of Internet protocol experts, including representatives of university and research labs, vendors, and government agencies. They were assembled primarily by the Host Requirements Working Group of the Internet Engineering Task Force (IETF).

The Editor of RFC-1122 and RFC-1123 (Bob Braden) acknowledges the tireless dedication of the following people, who attended many long meetings and generated 3 million bytes of electronic mail over an 18 month period in pursuit of these two documents: Philip Almquist, Dave Borman (Cray Research), Noel Chiappa, Dave Crocker (DEC), Steve Deering (Stanford), Mike Karels (Berkeley), Phil Karn (Bellcore), John Lekashman (NASA), Charles Lynn (BBN), Keith McCloghrie (TWG), Paul Mockapetris (ISI), Thomas Narten (Purdue), Craig Partridge (BBN), Drew Perkins (CMU), and James Van Bokkelen (FTP Software).

In addition, the following people made major contributions to the effort: Bill Barns (Mitre), Steve Bellovin (AT&T), Mike Brescia (BBN), Ed Cain (DCA), Annette DeSchon (ISI), Martin Gross (DCA), Phill Gross (NRI), Charles Hedrick (Rutgers), Van Jacobson (LBL), John Klensin (MIT), Mark Lottor (SRI), Milo Medin (NASA), Bill Melohn (Sun Microsystems), Greg Minshall (Kinetics), Jeff Mogul (DEC), John Mullen (CMC), Jon Postel (ISI), John Romkey (Epilogue Technology), and Mike StJohns (DCA). The following also made significant contributions to particular areas: Eric Allman (Berkeley), Rob Austein (MIT), Art Berggreen (ACC), Keith Bostic (Berkeley), Vint Cerf (NRI), Wayne Hathaway (NASA), Matt Korn (IBM), Erik Naggum (Naggum Software, Norway), Robert Ullmann (Prime Computer), David Waitzman (BBN), Frank Wancho (USA), Arun Welch (Ohio State), Bill Westfield (Cisco), and Rayan Zachariassen (Toronto).

The editors of this system (Joel Goldberger and Ken Berger of InfoMagic, Inc.) would like to thank everyone named above (including Bob Braden), and also the following individuals who provided both encouragement and assistance in the preparation of this system: Billy Brackenridge (The Voyager Company), Steve Knowles and Bruce Campbell (FTP Software), John Romkey (Epilogue Technologies), Vint Cerf (CNRI), Daniel Lynch (InterOp), and Zvi Alon (NetManage).

Link Layer Protocols

Introduction

To communicate on its directly-connected network, a host must implement the communication protocol used to interface to that network. We call this a link layer or media-access layer protocol.

There are a wide variety of link layer protocols, corresponding to the many different types of networks, several of which are mentioned below. The Address Resolution Protocol described in RFC-826, is a techique hosts and gateways **must** use to determine the mapping of Internet Addresses to Hardware Addresses. Trailer Encapsulations described in RFC-893 is an **optional** protocol hosts and gateways **may** negotiate to use for performance improvements.

Public data networks via X.25
ARPANET via 1822 LH, DH, or HDH
ARPANET via DDN Standard X.25
Ethernet and IEEE 802
NetBIOS Networks
Serial-Line Protocols

Public data networks via X.25

The formats specified for public data networks accessed via X.25 are described in RFC-877. Datagrams are transmitted over standard level-3 virtual circuits as complete packet sequences. Virtual circuits are usually established dynamically as required and time-out after a period of no traffic. Link-level retransmission, resequencing and flow control are performed by the network for each virtual circuit and by the LAPB link-level protocol. Note that a single X.25 virtual circuit may be used to multiplex all IP traffic between a pair of hosts. However, multiple parallel virtual circuits may be used in order to improve the utilization of the subscriber access line, in spite of small X.25 window sizes; this can result in random resequencing.

The correspondence between Internet and X.121 addresses is usually established by table-lookup. It is expected that this will be replaced by some sort of directory procedure in the future. The table of the hosts on the Public Data Network is in the Assigned Numbers.

The normal MTU is 576; however, the two DTE's (hosts or gateways) can use X.25 packet size negotiation to increase this value.

ARPANET via 1822 LH, DH, or HDH

The formats specified for ARPANET networks using 1822 access are described in BBN Report 1822 [3], which includes the procedures for several subscriber access methods. The Distant Host (DH) method is used when the host and IMP (the Defense Communication Agency calls it a Packet Switch Node or PSN) are separated by not more than about 2000 feet of cable, while the HDLC Distant Host (HDH) is used for greater distances where a modem is required. Under HDH, retransmission, resequencing and flow control are performed by the network and by the HDLC link-level protocol.

The IP encapsulation format is simply to include the IP datagram as the data portion of an 1822 message. In addition, the high-order 8 bits of the Message Id field (also known as the "link" field") should be set to 155. The MTU is 1007 octets.

The original IP address mapping (RFC-796) is in the process of being replaced by a new interface specification called AHIP-E; see RFC-1005 [61] for the proposal.

Gateways connected to ARPANET or MILNET IMPs using 1822 access must incorporate features to avoid host-port blocking (i.e., RFNM counting) and to detect and report as ICMP Unreachable messages the failure of destination hosts or gateways (i.e., convert the 1822 error messages to the appropriate ICMP messages).

In the development of a network interface it will be useful to review the IMP end-to-end protocol described in RFC-979.

ARPANET via DDN Standard X.25

The formats specified for ARPANET networks via X.25 are described in the Defense Data Network X.25 Host Interface Specification [6], which describes two sets of procedures: the DDN Basic X.25, and the DDN Standard X.25. Only DDN Standard X.25 provides the functionality required for interoperability assumptions of the Internet protocol.

The DDN Standard X.25 procedures are similar to the public data network X.25 procedures, except in the address mappings. Retransmission, resequencing and flowcontrol are performed by the network and by the LAPB link-level protocol. Multiple parallel virtual circuits may be used in order to improve the utilization of the subscriber access line; this can result in random resequencing.

Gateways connected to ARPANET or MILNET using Standard X.25 access must detect and report as ICMP Unreachable messages the failure of destination hosts or gateways (i.e., convert the X.25 diagnostic codes to the appropriate ICMP messages).

To achieve compatibility with 1822 interfaces, the effective MTU for a Standard X.25 interface is 1007 octets.

Serial-Line Protocols

In some configurations, gateways may be interconnected with each other by means of serial asynchronous or synchronous lines, with or without modems. When justified by the expected error rate and other factors, a link-level protocol may be required on the serial line. While there is no single Internet standard for this protocol, it is suggested that one of the following protocols be used.

- X.25 LAPB (Synchronous Lines)

  This is the link-level protocol used for X.25 network access. It includes HDLC "bit-stuffing" as well as rotating-window flow control and reliable delivery.

  A gateway must be configurable to play the role of either the DCE or the DTE.

- HDLC Framing (Synchronous Lines)

  This is just the bit-stuffing and framing rules of LAPB. It is the simplest choice, although it provides no flow control or reliable delivery; however, it does provide error detection.

- Xerox Synchronous Point-to-Point (Synchronous Lines)

  This Xerox protocol is an elaboration upon HDLC framing that includes negotiation of maximum packet sizes, dial-up or dedicated circuits, and half- or full-duplex operation [12].

- Serial Line IP (SLIP) (Asynchronous Lines)

  This protocol is included as part of the Berkeley and Sun UNIX Distributions, and is described in RFC-1055 "A Non-Standard for the Transmission of IP Datagrams over Serial Lines".

It will be important to make efficient use of the bandwidth available on a serial line between gateways. For example, it is desirable to provide some form of data compression. One possible standard compression algorithm is described if RFC-1144 "Compressing IP Headers for Low Speed Serial Lines". This and similar algorithms are tuned to the particular types of redundancy which occur in IP and TCP headers; however, more work is necessary to define a standard serial-line compression protocol for Internet gateways. Until a standard has been adopted, each vendor is free to choose a compression algorithm; of course, the result will only be useful on a serial line between two gateways using the same compression algorithm.

Another way to ensure maximum use of the bandwidth is to avoid unnecessary retransmissions at the link level. For some kinds of IP traffic, low delay is more important than reliable delivery. The serial line driver could distinguish such datagrams by their IP TOS field, and place them on a special high-priority, no-retransmission queue.

A serial point-to-point line between two gateways may be considered to be a (particularly simple) network, a "null net". Considered in this way, a serial linerequires no special considerations in the routing algorithms of the connected gateways, but does need an IP network number. To avoid the wholesale consumptionof Internet routing data-base space by null nets, we strongly recommend that subnetting be used for null net numbering, whenever possible.

For example, assume that network 128.203 is to be constructed of gateways joined by null nets; these nets are given (sub-)net numbers 128.203.1, 128.203.2, etc., and the two interfaces on each end of null net 128.203.s might have IP addresses 128.203.s.1 and 128.203.s.2.

An alternative model of a serial line is that it is not a network, but rather an internal communication path joining two "half gateways". It is possible to design an IGP and routing algorithm that treats a serial line in this manner [RFC-891, 52].

Ethernet and IEEE 802 Encapsulation

The IP encapsulation for Ethernets is described in RFC-894, while RFC-1042 describes the IP encapsulation for IEEE 802 networks.

Every Internet host connected to a 10Mbps Ethernet cable:

- **must** be able to send and receive packets using RFC-894 encapsulation;
- **should** be able to receive RFC-1042 packets, intermixed with RFC-894 packets; and
- **may** be able to send packets using RFC-1042 encapsulation.

An Internet host that implements sending both the RFC-894 and the RFC-1042 encapsulations **must** provide a configuration switch to select which is sent, and this switch **must** default to RFC- 894.

Note that the standard IP encapsulation in RFC-1042 does not use the protocol id value (K1=6) that IEEE reserved for IP; instead, it uses a value (K1=170) that implies an extension (the "SNAP") which can be used to hold the Ether-Type field. An Internet system **must not** send 802 packets using K1=6.

Address translation from Internet addresses to link-layer addresses on Ethernet and IEEE 802 networks **must** be managed by the Address Resolution Protocol (ARP).

The MTU for an Ethernet is 1500 and for 802.3 is 1492.

Discussion

The IEEE 802.3 specification provides for operation over a 10Mbps Ethernet cable, in which case Ethernet and IEEE 802.3 frames can be physically intermixed. A receiver can distinguish Ethernet and 802.3 frames by the value of the 802.3 Length field; this two-octet field coincides in the header with the Ether-Type field of an Ethernet frame. In particular, the 802.3 Length field must be less than or equal to 1500, while all valid Ether-Type values are greater than 1500.

Another compatibility problem arises with link-layer broadcasts. A broadcast sent with one framing will not be seen by hosts that can receive only the other framing.

The provisions of this section were designed to provide direct interoperation between 894-capable and 1042-capable systems on the same cable, to the maximum extent possible. It is intended to support the present situation where 894-only systems predominate, while providing an easy transition to a possible future in which 1042-capable systems become common.

Note that 894-only systems cannot interoperate directly with 1042-only systems. If the two system types are set up as two different logical networks on the same cable, they can communicate only through an IP gateway. Furthermore, it is not useful or even possible for a dual-format host to discover automatically which format to send, because of the problem of link-layer broadcasts.

ARP Packet Queue

The link layer **should** save (rather than discard) at least one (the latest) packet of each set of packets destined to the same unresolved IP address, and transmit the saved packet when the address has been resolved.

Discussion

Failure to follow this recommendation causes the first packet of every exchange to be lost. Although higher- layer protocols can generally cope with packet loss by retransmission, packet loss does impact performance. For example, loss of a TCP open request causes the initial round-trip time estimate to be inflated. UDP- based applications such as the Domain Name System are more seriously affected.

Link/Internet Layer Interface

The packet receive interface between the IP layer and the link layer **must** include a flag to indicate whether the incoming packet was addressed to a link-layer broadcast address.

Discussion
Although the IP layer does not generally know link layer addresses (since every different network medium typically has a different address format), the broadcast address on a broadcast-capable medium is an important special case.

The packet send interface between the IP and link layers must include the 5-bit TOS field.

The link layer must not report a Destination Unreachable error to IP solely because there is no ARP cache entry for a destination.

Internet Layer Protocols

Introduction

The Robustness Principle: "Be liberal in what you accept, and conservative in what you send" is particularly important in the Internet layer, where one misbehaving host can deny Internet service to many other hosts.

The protocol standards used in the Internet layer are:

- RFC-791 defines IP (the Internet Protocol) and gives an introduction to the architecture of the Internet.
- RFC-792 defines ICMP, (the Internet Control Message Protocol) which provides routing, diagnostic and error functionality for IP. Although ICMP messages are encapsulated within IP datagrams, ICMP processing is considered to be (and is typically implemented as) part of the IP layer.
- RFC-950 defines the mandatory subnet extension to the addressing architecture.
- The Internet Group Management Protocol IGMP, is part of a recommended extension to hosts and to the host-gateway interface to support Internet-wide multicasting at the IP level. This specification originally appeared as part of RFC-1112.

The target of an IP multicast may be an arbitrary group of Internet hosts. IP multicasting is designed as a natural extension of the link-layer multicasting facilities of some networks, and it provides a standard means for local access to such link-layer multicasting facilities.

Other important references are listed in References.

The Internet layer of host software **must** implement both IP and ICMP. See IP Multicasting for the requirements on support of IGMP.

The host IP layer has two basic functions: (1) choose the "next hop" gateway or host for outgoing IP datagrams and (2) reassemble incoming IP datagrams. The IP layer may also (3) implement intentional fragmentation of outgoing datagrams. Finally, the IP layer must (4) provide diagnostic and error functionality. We expect that IP layer functions may increase somewhat in the future, as further Internet control and management facilities are developed.

For normal datagrams, the processing is straightforward. For incoming datagrams, the IP layer:

(1) verifies that the datagram is correctly formatted;

(2) verifies that it is destined to the local host;

(3) processes options;

(4) reassembles the datagram if necessary; and

(5) passes the encapsulated message to the appropriate transport-layer protocol module.

For outgoing datagrams, the IP layer:

(1) sets any fields not set by the transport layer;

(2) selects the correct first hop on the connected network (a process called "routing");

(3) fragments the datagram if necessary and if intentional fragmentation is

implemented; and

(4) passes the packet(s) to the appropriate link-layer driver.

A host is said to be multihomed if it has multiple IP addresses. Multihoming introduces considerable confusion and complexity into the protocol suite, and it is an area in which the Internet architecture falls seriously short of solving all problems. There are two distinct problem areas in multihoming:

(1) Local multihoming -- the host itself is multihomed; or

(2) Remote multihoming -- the local host needs to communicate with a remote multihomed host.

At present, remote multihoming **must** be handled at the application layer. A host **may** support local multihoming, which is discussed in this document, and in particular in Local Multihoming.

Any host that forwards datagrams generated by another host is acting as a gateway and **must** also meet the specifications laid out in the gateway requirements RFC [RFC-1009]. An Internet host that includes embedded gateway code **must** have a configuration switch to disable the gateway function, and this switch **must** default to the non-gateway mode. In this mode, a datagram arriving through one interface will not be forwarded to another host or gateway (unless it is source-routed), regardless of whether the host is single- homed or multihomed. The host software **must not** automatically move into gateway mode if the host has more than one interface, as the operator of the machine may neither want to provide that service nor be competent to do so.

In the following, the action specified in certain cases is to "silently discard" a received datagram. This means that the datagram will be discarded without further processing and that the host will not send any ICMP error message as a result. However, for diagnosis of problems a host **should** provide the capability of logging the error, including the contents of the silently-discarded datagram, and **should** record the event in a statistics counter.

Silent discard of erroneous datagrams is generally intended to prevent "broadcast storms".

Internet Control Message Protocol Summary

ICMP is specified in RFC-792.

If an ICMP message of unknown type is received, it **must** be silently discarded.

Every ICMP error message includes the Internet header and at least the first 8 data octets of the datagram that triggered the error; more than 8 octets **may** be sent; this header and data **must** be unchanged from the received datagram.

In those cases where the Internet layer is required to pass an ICMP error message to the transport layer, the IP protocol number **must** be extracted from the original header and used to select the appropriate transport protocol entity to handle the error.

An ICMP error message **should** be sent with normal (i.e., zero) TOS bits.

An ICMP error message **must not** be sent as the result of receiving:

* an ICMP error message, or
* a datagram destined to an IP broadcast or IP multicast address, or
* a datagram sent as a link-layer broadcast, or
* a non-initial fragment, or
* a datagram whose source address does not define a single host -- e.g., a zero address, a loopback address, a broadcast address, a multicast address, or a Class E address.

NOTE: THESE RESTRICTIONS TAKE PRECEDENCE OVER ANY REQUIREMENT ELSEWHERE IN THIS DOCUMENT FOR SENDING ICMP ERROR MESSAGES.

These rules will prevent the "broadcast storms" that have resulted from hosts returning ICMP error messages in response to broadcast datagrams. For example, a broadcast UDP segment to a non-existent port could trigger a flood of ICMP Destination Unreachable datagrams from all machines that do not have a client for that destination port. On a large Ethernet, the resulting collisions can render the network useless for a second or more.

Every datagram that is broadcast on the connected network should have a valid IP broadcast address as its IP destination. However, some hosts violate this rule. To be certain to detect broadcast datagrams, therefore, hosts are required to check for a link-layer broadcast as well as an IP-layer broadcast address.

This requires that the link layer inform the IP layer when a link-layer broadcast datagram has been received; see Link/Internet Layer Interface.

IP Multicasting

A host **should** support local IP multicasting on all connected networks for which a mapping from Class D IP addresses to link-layer addresses has been specified (see below). Support for local IP multicasting includes sending multicast datagrams, joining multicast groups and receiving multicast datagrams, and leaving multicast groups. This implies support for all of RFC-1112, "Host Extensions for IP Multicasting" except the IGMP protocol itself, which is **optional**.

> **Discussion**
> IGMP provides gateways that are capable of multicast routing with the information required to support IP multicasting across multiple networks. At this time, multicast-routing gateways are in the experimental stage and are not widely available. For hosts that are not connected to networks with multicast-routing gateways or that do not need to receive multicast datagrams originating on other networks, IGMP serves no purpose and is therefore optional for now. However, the rest of RFC-1112 is currently recommended for the purpose of providing IP-layer access to local network multicast addressing, as a preferable alternative to local broadcast addressing. It is expected that IGMP will become recommended at some future date, when multicast-routing gateways have become more widely available.

If IGMP is not implemented, a host **should** still join the "all- hosts" group (224.0.0.1) when the IP layer is initialized and remain a member for as long as the IP layer is active.

> **Discussion**
> Joining the "all-hosts" group will support strictly local uses of multicasting, e.g., a gateway discovery protocol, even if IGMP is not implemented.
>
> The mapping of IP Class D addresses to local addresses is currently specified for the following types of networks:
>
> - Ethernet/IEEE 802.3, as defined in RFC-1112.
> - Any network that supports broadcast but not multicast, addressing: all IP Class D addresses map to the local broadcast address.
> - Any type of point-to-point link (e.g., SLIP or HDLC links): no mapping required. All IP multicast datagrams are sent as-is, inside the local framing.
>
> Mappings for other types of networks will be specified in the future.
>
> A host **should** provide a way for higher-layer protocols or applications to determine which of the host's connected network(s) support IP multicast addressing.

Error Reporting

Wherever practical, hosts **must** return ICMP error datagrams on detection of an error, except in those cases where returning an ICMP error message is specifically prohibited.

> **Discussion**
> A common phenomenon in datagram networks is the "black hole disease": datagrams are sent out, but nothing comes back. Without any error datagrams, it is difficult for the user to figure out what the problem is.

Internet/Transport Layer Interface

The interface between the IP layer and the transport layer **must** provide full access to all the mechanisms of the IP layer, including options, Type-of-Service, and Time-to-Live. The transport layer **must** either have mechanisms to set these interface parameters, or provide a path to pass them through from an application, or both.

> **Discussion**
> Applications are urged to make use of these mechanisms where applicable, even when the mechanisms are not currently effective in the Internet (e.g., TOS). This will allow these mechanisms to be immediately useful when they do become effective, without a large amount of retrofitting of host software.

We now describe a conceptual interface between the transport layer and the IP layer, as a set of procedure calls. This is an extension of the information in IP Interfaces.

* **Send Datagram**

  SEND(src, dst, prot, TOS, TTL, BufPTR, len, Id, DF, opt => result)

  where the parameters are defined in RFC-791. Passing an Id parameter is optional; see Identification.

* **Receive Datagram**

  RECV(BufPTR, prot => result, src, dst, SpecDest, TOS, len, opt)

  All the parameters are defined in RFC-791, except for:

  SpecDest = specific-destination address of datagram (defined in Addressing)

  The result parameter dst contains the datagram's destination address. Since this may be a broadcast or multicast address, the SpecDest parameter (not originally shown in RFC-791) **must** be passed. The parameter opt contains all the IP options received in the datagram; these **must** also be passed to the transport layer.

* **Select Source Address**

  GET_SRCADDR(remote, TOS) -> local

  remote = remote IP address
  TOS = Type-of-Service
  local = local IP address

* **Find Maximum Datagram Sizes**

  GET_MAXSIZES(local, remote, TOS) -> MMS_R, MMS_S

  MMS_R = maximum receive transport-message size.
  MMS_S = maximum send transport-message size.
  (local, remote, TOS defined above)

* **Advice on Delivery Success**

  ADVISE_DELIVPROB(sense, local, remote, TOS)

  Here the parameter sense is a 1-bit flag indicating whether positive or negative advice is being given; see the discussion in Dead Gateway Detection. The other parameters were defined earlier.

* **Send ICMP Message**

  SEND_ICMP(src, dst, TOS, TTL, BufPTR, len, Id, DF, opt) -> result

(Parameters defined in RFC-791 and RFC-792).

Passing an Id parameter is optional; see Identification. The transport layer **must** be able to send certain ICMP messages: Port Unreachable or any of the query-type messages. This function could be considered to be a special case of the SEND() call, of course; we describe it separately for clarity.

* **Receive ICMP Message**

RECV_ICMP(BufPTR) -> result, src, dst, len, opt

The IP layer **must** pass certain ICMP messages up to the appropriate transport-layer routine. This function could be considered to be a special case of the RECV() call, of course; we describe it separately for clarity.

For an ICMP error message, the data that is passed up **must** include the original Internet header plus all the octets of the original message that are included in the ICMP message. This data will be used by the transport layer to locate the connection state information, if any.

In particular, the following ICMP messages are to be passed up:

- Destination Unreachable
- Source Quench
- Echo Reply (to ICMP user interface, unless the Echo Request originated in the IP layer)
- Timestamp Reply (to ICMP user interface)
- Time Exceeded

Discussion

In the future, there may be additions to this interface to pass path data (see Route Cache) between the IP and transport layers.

User Datagram Protocol -- UDP
Introduction

The User Datagram Protocol (UDP),defined in RFC-768, offers only a minimal transport service -- non-guaranteed datagram delivery -- and gives applications direct access to the datagram service of the IP layer. UDP is used by applications that do not require the level of service of TCP or that wish to use communications services (e.g., multicast or broadcast delivery) not available from TCP.

UDP is almost a null protocol; the only services it provides over IP are checksumming of data and multiplexing by port number. Therefore, an application program running over UDP must deal directly with end-to-end communication problems that a connection-oriented protocol would have handled -- e.g., retransmission for reliable delivery, packetization and reassembly, flow control, congestion avoidance, etc., when these are required. The fairly complex coupling between IP and TCP will be mirrored in the coupling between UDP and many applications using UDP.

Transmission Control Protocol -- TCP
Introduction

The Transmission Control Protocol TCP is defined in RFC-793 is the primary virtual-circuit transport protocol for the Internet suite. TCP provides reliable, in-sequence delivery of a full-duplex stream of octets (8-bit bytes). TCP is used by those applications needing reliable, connection-oriented transport service, e.g., mail (SMTP), file transfer (FTP), and virtual terminal service (Telnet).

TCP -- Specific Issues

When to Send a TCP ACK Segment

A host that is receiving a stream of TCP data segments can increase efficiency in both the Internet and the hosts by sending fewer than one ACK (acknowledgment) segment per data segment received; this is known as a "delayed ACK" and is described in RFC-813 "Window and Acknowledgement Strategy in TCP.

A TCP **should** implement a delayed ACK, but an ACK should not be excessively delayed; in particular, the delay **must** be less than 0.5 seconds, and in a stream of full-sized segments there **should** be an ACK for at least every second segment.

Discussion

A delayed ACK gives the application an opportunity to update the window and perhaps to send an immediate response. In particular, in the case of character-mode remote login, a delayed ACK can reduce the number of segments sent by the server by a factor of 3 (ACK, window update, and echo character all combined in one segment).

In addition, on some large multi-user hosts, a delayed ACK can substantially reduce protocol processing overhead by reducing the total number of packets to be processed. However, excessive delays on ACK's can disturb the round-trip timing and packet "clocking" algorithms [TCP:7].

When to Send a TCP Window Update

A TCP **must** include a SWS avoidance algorithm in the receiver as described in RFC-813 "Window and Acknowledgement Strategy in TCP".

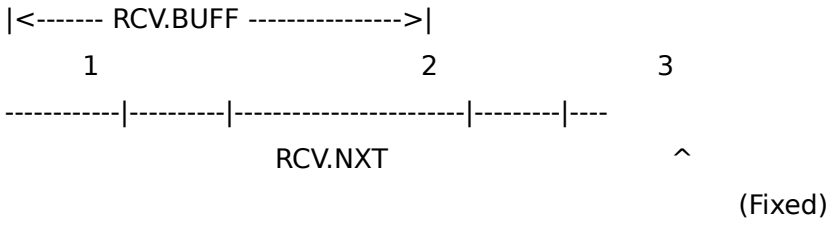
Implementation

The receiver's SWS avoidance algorithm determines when the right window edge may be advanced; this is customarily known as "updating the window". This algorithm combines with the delayed ACK algorithm (see When to Send an ACK Segment) to determine when an ACK segment containing the current window will really be sent to the receiver. We use the notation of RFC-793; see Figures 4 and 5 in that document.

The solution to receiver SWS is to avoid advancing the right window edge RCV.NXT+RCV.WND in small increments, even if data is received from the network in small segments.

Suppose the total receive buffer space is RCV.BUFF. At any given moment, RCV.USER octets of this total may be tied up with data that has been received and acknowledged but which the user process has not yet consumed. When the connection is quiescent, RCV.WND = RCV.BUFF and RCV.USER = 0.

Keeping the right window edge fixed as data arrives and is acknowledged requires that the receiver offer less than its full buffer space, i.e., the receiver must specify a RCV.WND that keeps RCV.NXT+RCV.WND constant as RCV.NXT increases. Thus, the total buffer space RCV.BUFF is generally divided into three parts:



1 - RCV.USER = data received but not yet consumed;

2 - RCV.WND = space advertised to sender;

3 - Reduction = space available but not yet advertised.

The suggested SWS avoidance algorithm for the receiver is to keep RCV.NXT+RCV.WND fixed until the reduction satisfies:

RCV.BUFF - RCV.USER - RCV.WND >= min(Fr * RCV.BUFF, Eff.snd.MSS)

where Fr is a fraction whose recommended value is 1/2, and Eff.snd.MSS is the effective send MSS for the connection (see Maximum Segement Size). When the inequality is satisfied, RCV.WND is set to RCV.BUFF-RCV.USER.

Note that the general effect of this algorithm is to advance RCV.WND in increments of Eff.snd.MSS (for realistic receive buffers: Eff.snd.MSS < RCV.BUFF/2). Note also that the receiver must use its own Eff.snd.MSS, assuming it is the same as the sender's.

When to Send TCP Data

A TCP **must** include a SWS avoidance algorithm in the sender.

A TCP **should** implement the Nagle Algorithm as described in RFC-896 "Congestion Control in IP/TCP" to coalesce short segments. However, there **must** be a way for an application to disable the Nagle algorithm on an individual connection. In all cases, sending data is also subject to the limitation imposed by the Slow Start algorithm (Retransmission Timeout).

Discussion

The Nagle algorithm is generally as follows:

> If there is unacknowledged data (i.e., SND.NXT > SND.UNA), then the sending TCP buffers all user data (regardless of the PSH bit), until the outstanding data has been acknowledged or until the TCP can send a full-sized segment (Eff.snd.MSS bytes; see Maximum Segment Size).

Some applications (e.g., real-time display window updates) require that the Nagle algorithm be turned off, so small data segments can be streamed out at the maximum rate. (See Implementation).

When to Send TCP Data Implementation

The sender's SWS avoidance algorithm is more difficult than the receivers's, because the sender does not know (directly) the receiver's total buffer space RCV.BUFF. An approach which has been found to work well is for the sender to calculate Max(SND.WND), the maximum send window it has seen so far on the connection, and to use this value as an estimate of RCV.BUFF. Unfortunately, this can only be an estimate; the receiver may at any time reduce the size of RCV.BUFF. To avoid a resulting deadlock, it is necessary to have a timeout to force transmission of data, overriding the SWS avoidance algorithm. In practice, this timeout should seldom occur.

The "useable window" (as discussed the Silly Window Syndrome section of RFC-813) is:

U = SND.UNA + SND.WND - SND.NXT

i.e., the offered window less the amount of data sent but not acknowledged. If D is the amount of data queued in the sending TCP but not yet sent, then the following set of rules is recommended.

Send data:

(1) if a maximum-sized segment can be sent, i.e, if:
min(D,U) >= Eff.snd.MSS;

(2) or if the data is pushed and all queued data can be sent now, i.e., if:
[SND.NXT = SND.UNA and] PUSHED,and
D <= U

(the bracketed condition is imposed by the Nagle algorithm);

(3) or if at least a fraction Fs of the maximum window can be sent, i.e., if:
[SND.NXT = SND.UNA], and
min(D.U) >= Fs * Max(SND.WND);

(4) or if data is PUSHed and the override timeout occurs.

Here Fs is a fraction whose recommended value is 1/2. The override timeout should be in the range 0.1 - 1.0 seconds. It may be convenient to combine this timer with the timer used to probe zero windows (Probing Zero Windows).

Finally, note that the SWS avoidance algorithm just specified is to be used instead of the sender-side algorithm contained in RFC-813 "Window and Acknowledgement Strategy in TCP.

Connection Failures

Excessive retransmission of the same segment by TCP indicates some failure of the remote host or the Internet path. This failure may be of short or long duration. The following procedure **must** be used to handle excessive retransmissions of data segments (See RFC-816 "Fault Isolation and Recovery":

(a) There are two thresholds R1 and R2 measuring the amount of retransmission that has occurred for the same segment. R1 and R2 might be measured in time units or as a count of retransmissions.

(b) When the number of transmissions of the same segment reaches or exceeds threshold R1, pass negative advice to the IP layer, to trigger dead-gateway diagnosis.

(c) When the number of transmissions of the same segment reaches a threshold R2 greater than R1, close the connection.

(d) An application **must** be able to set the value for R2 for a particular connection. For example, an interactive application might set R2 to "infinity," giving the user control over when to disconnect.

(e) TCP **should** inform the application of the delivery problem (unless such information has been disabled by the application; see Asynchronous Reports), when R1 is reached and before R2. This will allow a remote login (User Telnet) application program to inform the user, for example.

The value of R1 **should** correspond to at least 3 retransmissions, at the current RTO. The value of R2 **should** correspond to at least 100 seconds.

An attempt to open a TCP connection could fail with excessive retransmissions of the SYN segment or by receipt of a RST segment or an ICMP Port Unreachable. SYN retransmissions **must** be handled in the general way just described for data retransmissions, including notification of the application layer.

However, the values of R1 and R2 may be different for SYN and data segments. In particular, R2 for a SYN segment **must** be set large enough to provide retransmission of the segment for at least 3 minutes. The application can close the connection (i.e., give up on the open attempt) sooner, of course.

Discussion

Some Internet paths have significant setup times, and the number of such paths is likely to increase in the future.

TCP Keep-Alives

Implementors **may** include "keep-alives" in their TCP implementations, although this practice is not universally accepted. If keep-alives are included, the application **must** be able to turn them on or off for each TCP connection, and they **must** default to off.

Keep-alive packets **must** only be sent when no data or acknowledgement packets have been received for the connection within an interval. This interval **must** be configurable and **must** default to no less than two hours.

It is extremely important to remember that ACK segments that contain no data are not reliably transmitted by TCP. Consequently, if a keep-alive mechanism is implemented it **must not** interpret failure to respond to any specific probe as a dead connection.

An implementation **should** send a keep-alive segment with no data; however, it **may** be configurable to send a keep-alive segment containing one garbage octet, for compatibility with erroneous TCP implementations. (See Discussion).

TCP Keep-Alives Discussion

A "keep-alive" mechanism periodically probes the other end of a connection when the connection is otherwise idle, even when there is no data to be sent. The TCP specification does not include a keep-alive mechanism because it could: (1) cause perfectly good connections to break during transient Internet failures; (2) consume unnecessary bandwidth ("if no one is using the connection, who cares if it is still good?"); and (3) cost money for an Internet path that charges for packets.

Some TCP implementations, however, have included a keep-alive mechanism. To confirm that an idle connection is still active, these implementations send a probe segment designed to elicit a response from the peer TCP. Such a segment generally contains SEG.SEQ = SND.NXT-1 and may or may not contain one garbage octet of data. Note that on a quiet connection SND.NXT = RCV.NXT, so that this SEG.SEQ will be outside the window. Therefore, the probe causes the receiver to return an acknowledgment segment, confirming that the connection is still live. If the peer has dropped the connection due to a network partition or a crash, it will respond with a RST instead of an acknowledgment segment.

Unfortunately, some misbehaved TCP implementations fail to respond to a segment with SEG.SEQ = SND.NXT-1 unless the segment contains data. Alternatively, an implementation could determine whether a peer responded correctly to keep-alive packets with no garbage data octet.

A TCP keep-alive mechanism should only be invoked in server applications that might otherwise hang indefinitely and consume resources unnecessarily if a client crashes or aborts a connection during a network failure.

TCP Multihoming Issues

If an application on a multihomed host does not specify the local IP address when actively opening a TCP connection, then the TCP **must** ask the IP layer to select a local IP address before sending the (first) SYN. See the function GET_SRCADDR() in Internet/Transport Layer Interface.

At all other times, a previous segment has either been sent or received on this connection, and TCP **must** use the same local address is used that was used in those previous segments.

IP Options

When received options are passed up to TCP from the IP layer, TCP **must** ignore options that it does not understand.

A TCP **may** support the Time Stamp and Record Route options.

An application **must** be able to specify a source route when it actively opens a TCP connection, and this **must** take precedence over a source route received in a datagram.

When a TCP connection is OPENed passively and a packet arrives with a completed IP Source Route option (containing a return route), TCP **must** save the return route and use it for all segments sent on this connection. If a different source route arrives in a later segment, the later definition **should** override the earlier one.

ICMP Messages

TCP **must** act on an ICMP error message passed up from the IP layer, directing it to the connection that created the error. The necessary demultiplexing information can be found in the IP header contained within the ICMP message.

- Source Quench

  TCP **must** react to a Source Quench by slowing transmission on the connection. The RECOMMENDED procedure is for a Source Quench to trigger a "slow start," as if a retransmission timeout had occurred.

- Destination Unreachable -- codes 0, 1, 5

  Since these Unreachable messages indicate soft error conditions, TCP **must not** abort the connection, and it **should** make the information available to the application.

  Discussion

  TCP could report the soft error condition directly to the application layer with an upcall to the ERROR_REPORT routine, or it could merely note the message and report it to the application only when and if the TCP connection times out.

- Destination Unreachable -- codes 2-4

  These are hard error conditions, so TCP **should** abort the connection.

- Time Exceeded -- codes 0, 1

  This should be handled the same way as Destination Unreachable codes 0, 1, 5 (see above).

- Parameter Problem

  This should be handled the same way as Destination Unreachable codes 0, 1, 5 (see above).

Remote Address Validation

A TCP implementation **must** reject as an error a local OPEN call for an invalid remote IP address (e.g., a broadcast or multicast address).

An incoming SYN with an invalid source address **must** be ignored either by TCP or by the IP layer.

A TCP implementation **must** silently discard an incoming SYN segment that is addressed to a broadcast or multicast address.

TCP Traffic Patterns

Implementation
The TCP protocol specification RFC-793 gives the implementor much freedom in designing the algorithms that control the message flow over the connection -- packetizing, managing the window, sending acknowledgments, etc. These design decisions are difficult because a TCP must adapt to a wide range of traffic patterns. Experience has shown that a TCP implementor needs to verify the design on two extreme traffic patterns:

- Single-character Segments

  Even if the sender is using the Nagle Algorithm, when a TCP connection carries remote login traffic across a low-delay LAN the receiver will generally get a stream of single-character segments. If remote terminal echo mode is in effect, the receiver's system will generally echo each character as it is received.

- Bulk Transfer

  When TCP is used for bulk transfer, the data stream should be made up (almost) entirely of segments of the size of the effective MSS. Although TCP uses a sequence number space with byte (octet) granularity, in bulk-transfer mode its operation should be as if TCP used a sequence space that counted only segments.

Experience has furthermore shown that a single TCP can effectively and efficiently handle these two extremes.

The most important tool for verifying a new TCP implementation is a packet trace program. There is a large volume of experience showing the importance of tracing a variety of traffic patterns with other TCP implementations and studying the results carefully.

TCP Efficiency

Implementation
Extensive experience has led to the following suggestions for efficient implementation of TCP:

(a) Don't Copy Data

In bulk data transfer, the primary CPU-intensive tasks are copying data from one place to another and checksumming the data. It is vital to minimize the number of copies of TCP data. Since the ultimate speed limitation may be fetching data across the memory bus, it may be useful to combine the copy with checksumming, doing both with a single memory fetch.

(b) Hand-Craft the Checksum Routine

A good TCP checksumming routine is typically two to five times faster than a simple and direct implementation of the definition. Great care and clever coding are often required and advisable to make the checksumming code "blazing fast". See RFC-1071 "Computing the Internet Checksum".

(c) Code for the Common Case

TCP protocol processing can be complicated, but for most segments there are only a few simple decisions to be made. Per-segment processing will be greatly speeded up by coding the main line to minimize the number of decisions in the most common case.

TCP/Application Layer Interface

Asynchronous Reports
Type-of-Service
Flush Call
Multihoming

TCP Asynchronous Reports

There **must** be a mechanism for reporting soft TCP error conditions to the application. Generically, we assume this takes the form of an application-supplied ERROR_REPORT routine that may be upcalled [INTRO:7] asynchronously from the transport layer:

ERROR_REPORT(local connection name, reason, subreason)

The precise encoding of the reason and subreason parameters is not specified here. However, the conditions that are reported asynchronously to the application **must** include:

* ICMP error message arrived (ICMP Messages)

* Excessive retransmissions (Connection Failures)

* Urgent pointer advance (Urgent Pointer).

However, an application program that does not want to receive such ERROR_REPORT calls **should** be able to effectively disable these calls.

Discussion

These error reports generally reflect soft errors that can be ignored without harm by many applications. It has been suggested that these error report calls should default to "disabled," but this is not required.

TCP Type-of-Service

The application layer **must** be able to specify the Type-of- Service (TOS) for segments that are sent on a connection. It not required, but the application **should** be able to change the TOS during the connection lifetime. TCP **should** pass the current TOS value without change to the IP layer, when it sends segments on the connection.

The TOS will be specified independently in each direction on the connection, so that the receiver application will specify the TOS used for ACK segments.

TCP **may** pass the most recently received TOS up to the application.

Discussion

Some applications (e.g., SMTP) change the nature of their communication during the lifetime of a connection, and therefore would like to change the TOS specification.

Note also that the OPEN call specified in RFC-793 includes a parameter ("options") in which the caller can specify IP options such as source route, record route, or timestamp.

TCP Flush Call

Some TCP implementations have included a FLUSH call, which will empty the TCP send queue of any data for which the user has issued SEND calls but which is still to the right of the current send window. That is, it flushes as much queued send data as possible without losing sequence number synchronization. This is useful for implementing the "abort output" function of Telnet.

TCP Multihoming

The user interface outlined in sections 2.7 and 3.8 of RFC- 793 needs to be extended for multihoming. The OPEN call **must** have an optional parameter:

OPEN(... [local IP address,] ...)

to allow the specification of the local IP address.

Discussion

Some TCP-based applications need to specify the local IP address to be used to open a particular connection; FTP is an example.

Implementation

A passive OPEN call with a specified "local IP address" parameter will await an incoming connection request to that address. If the parameter is unspecified, a passive OPEN will await an incoming connection request to any local IP address, and then bind the local IP address of the connection to the particular address that is used.

For an active OPEN call, a specified "local IP address" parameter will be used for opening the connection. If the parameter is unspecified, the networking software will choose an appropriate local IP address (see Multihoming Requirements) for the connection.

Silly Window Syndrome (SWS): a stable pattern of small incremental window movements resulting in extremely poor TCP performance.

[INTRO:1] "Requirements for Internet Hosts -- Application and Support," IETF Host Requirements Working Group, R. Braden, Ed., **RFC-1123**, October 1989.

[INTRO:2] "Requirements for Internet Gateways," R. Braden and J. Postel, **RFC-1009**, June 1987.

[INTRO:3] "DDN Protocol Handbook," **NIC-50004**, **NIC-50005**, **NIC-50006**, (three volumes), SRI International, December 1985.

[INTRO:4] "Official Internet Protocols," J. Reynolds and J. Postel, **RFC-1011**, May 1987.

This document is republished periodically with new RFC numbers; the latest version must be used.

[INTRO:5] "Protocol Document Order Information," O. Jacobsen and J. Postel, **RFC-980**, March 1986.

[INTRO:6] "Assigned Numbers," J. Reynolds and J. Postel, **RFC-1060**, March 1990.

This document is republished periodically with new RFC numbers; the latest version must be used.

[INTRO:7] "Modularity and Efficiency in Protocol Implementations," D. Clark, **RFC-817**, July 1982.

[INTRO:8] "The Structuring of Systems Using Upcalls," D. Clark, 10th ACM SOSP, Orcas Island, Washington, December 1985.

[INTRO:9] "A Protocol for Packet Network Intercommunication," V. Cerf and R. Kahn, IEEE Transactions on Communication, May 1974.

[INTRO:10] "The ARPA Internet Protocol," J. Postel, C. Sunshine, and D. Cohen, Computer Networks, Vol. 5, No. 4, July 1981.

[INTRO:11] "The DARPA Internet Protocol Suite," B. Leiner, J. Postel, R. Cole and D. Mills, Proceedings INFOCOM 85, IEEE, Washington DC, March 1985. Also in: IEEE Communications Magazine, March 1985. Also available as **ISI-RS-85-153**.

[INTRO:12] "Final Text of DIS8473, Protocol for Providing the Connectionless Mode Network Service," ANSI, published as **RFC-994**, March 1986.

[INTRO:13] "End System to Intermediate System Routing Exchange Protocol," ANSI X3S3.3, published as **RFC-995**, April 1986.

[LINK:1] "Trailer Encapsulations," S. Leffler and M. Karels, **RFC-893**, April 1984.

[LINK:2] "An Ethernet Address Resolution Protocol," D. Plummer, **RFC-826**, November 1982.

[LINK:3] "A Standard for the Transmission of IP Datagrams over Ethernet Networks," C. Hornig, **RFC-894**, April 1984.

[LINK:4] "A Standard for the Transmission of IP Datagrams over IEEE 802 "Networks," J. Postel and J. Reynolds, **RFC-1042**, February 1988.

This RFC contains a great deal of information of importance to Internet implementers planning to use IEEE 802 networks.

[IP:1] "Internet Protocol (IP)," J. Postel, **RFC-791**, September 1981.

[IP:2] "Internet Control Message Protocol (ICMP)," J. Postel, **RFC-792**, September 1981.

[IP:3] "Internet Standard Subnetting Procedure," J. Mogul and J. Postel, **RFC-950**, August 1985.

[IP:4] "Host Extensions for IP Multicasting," S. Deering, **RFC-1112**, August 1989.

[IP:5] "Military Standard Internet Protocol," **MIL-STD-1777**, Department of Defense, August 1983.

This specification, as amended by RFC-963, is intended to describe the Internet Protocol but has some serious omissions (e.g., the mandatory subnet extension [IP:3] and the optional multicasting extension [IP:4]). It is also out of date. If there is a conflict, RFC-791, RFC-792, and RFC-950 must be taken as authoritative, while the present document is authoritative over all.

[IP:6] "Some Problems with the Specification of the Military Standard Internet Protocol," D. Sidhu, **RFC-963**, November 1985.

[IP:7] "The TCP Maximum Segment Size and Related Topics," J. Postel, **RFC-879**, November 1983.

Discusses and clarifies the relationship between the TCP Maximum Segment Size option and the IP datagram size.

[IP:8] "Internet Protocol Security Options," B. Schofield, **RFC-1108**, October 1989.

[IP:9] "Fragmentation Considered Harmful," C. Kent and J. Mogul, ACM SIGCOMM-87, August 1987. Published as ACM Comp Comm Review, Vol. 17, no. 5.

This useful paper discusses the problems created by Internet fragmentation and presents alternative solutions.

[IP:10] "IP Datagram Reassembly Algorithms," D. Clark, **RFC-815**, July 1982.

This and the following paper should be read by every implementor.

[IP:11] "Fault Isolation and Recovery," D. Clark, **RFC-816**, July 1982.

[IP:12] "Broadcasting Internet Datagrams in the Presence of Subnets," J. Mogul, **RFC-922**, October 1984.

[IP:13] "Name, Addresses, Ports, and Routes," D. Clark, **RFC-814**, July 1982.

[IP:14] "Something a Host Could Do with Source Quench: The Source Quench Introduced Delay (SQUID)," W. Prue and J. Postel, **RFC-1016**, July 1987.

This RFC first described directed broadcast addresses. However, the bulk of the RFC is concerned with gateways, not hosts.

[UDP:1] "User Datagram Protocol," J. Postel, **RFC-768**, August 1980.

[TCP:1] "Transmission Control Protocol," J. Postel, **RFC-793**, September 1981.

[TCP:2] "Transmission Control Protocol," **MIL-STD-1778**, US Department of Defense, August 1984.

This specification as amended by RFC-964 is intended to describe the same protocol as RFC-793 [TCP:1]. If there is a conflict, RFC-793 takes precedence, and the present document is authoritative over both.

[TCP:3] "Some Problems with the Specification of the Military Standard Transmission Control Protocol," D. Sidhu and T. Blumer, **RFC-964**, November 1985.

[TCP:4] "The TCP Maximum Segment Size and Related Topics," J. Postel, **RFC-879**, November 1983.

[TCP:5] "Window and Acknowledgment Strategy in TCP," D. Clark, **RFC-813**, July 1982.

[TCP:6] "Round Trip Time Estimation," P. Karn & C. Partridge, ACM SIGCOMM-87, August 1987.

[TCP:7] "Congestion Avoidance and Control," V. Jacobson, ACM SIGCOMM-88, August 1988.

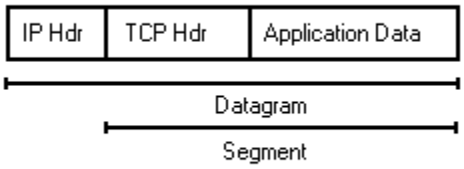
[TCP:8] "Modularity and Efficiency in Protocol Implementation," D. Clark, **RFC-817**, July 1982.

[TCP:9] "Congestion Control in IP/TCP," J. Nagle, **RFC-896**, January 1984.

[TCP:10] "Computing the Internet Checksum," R. Braden, D. Borman, and C. Partridge, **RFC-1071**, September 1988.

[TCP:11] "TCP Extensions for Long-Delay Paths," V. Jacobson & R. Braden, **RFC-1072**, October 1988.

A segment is the unit of end-to-end transmission in the TCP protocol. A segment consists of a TCP header followed by application data. A segment is transmitted by encapsulation inside an IP datagram.

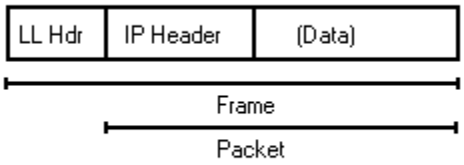


In this description of the lower-layer protocols, a message is the unit of transmission in a transport layer protocol. In particular, a TCP segment is a message. A message consists of a transport protocol header followed by application protocol data. To be transmitted end-to-end through the Internet, a message must be encapsulated inside a datagram.

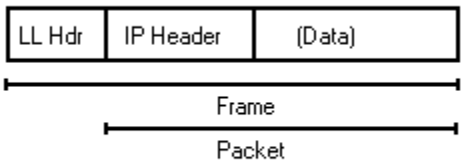
An IP datagram is the unit of end-to-end transmission in the IP protocol. An IP datagram consists of an IP header followed by transport layer data, i.e., of an IP header followed by a message.

In the description of the internet layer, the unqualified term "datagram" should be understood to refer to an IP datagram.

A packet is the unit of data passed across the interface between the internet layer and the link layer. It includes an IP header and data. A packet may be a complete IP datagram or a fragment of an IP datagram.



A frame is the unit of transmission in a link layer protocol, and consists of a link-layer header followed by a packet.



A network to which a host is interfaced is often known as the "local network" or the "subnetwork" relative to that host. However, these terms can cause confusion, and therefore we use the term "connected network" in this document.

A host is said to be multihomed if it has multiple IP addresses.

This is a physical interface to a connected network and has a (possibly unique) link-layer address. Multiple physical network interfaces on a single host may share the same link-layer address, but the address must be unique for different hosts on the same physical network.

We define a logical [network] interface to be a logical path, distinguished by a unique IP address, to a connected network.

This is the effective destination address of a datagram, even if it is broadcast or multicast.

At a given moment, all the IP datagrams from a particular source host to a particular destination host will typically traverse the same sequence of gateways. We use the term "path" for this sequence. Note that a path is uni-directional; it is not unusual to have different paths in the two directions between a given host pair.

The maximum transmission unit, i.e., the size of the largest packet that can be transmitted.

File Transfer Protocol
FTP

Introduction

The File Transfer Protocol FTP is the primary Internet standard for file transfer. The current specification is contained in RFC-959.

FTP uses separate simultaneous TCP connections for control and for data transfer. The FTP protocol includes many features, some of which are not commonly implemented. However, for every feature in FTP, there exists at least one implementation.

Internet users have been unnecessarily burdened for years by deficient FTP implementations. Protocol implementors have suffered from the erroneous opinion that implementing FTP ought to be a small and trivial task. This is wrong, because FTP has a user interface, because it has to deal (correctly) with the whole variety of communication and operating system errors that may occur, and because it has to handle the great diversity of real file systems in the world.

Specific Issues
FTP/User Interface

Specific Issues

Non-standard Command Verbs

FTP allows "experimental" commands, whose names begin with "X". If these commands are subsequently adopted as standards, there may still be existing implementations using the "X" form. At present, this is true for the directory commands:

RFC-959	**"Experimental"**
MKD	XMKD
RMD	XRMD
PWD	XPWD
CDUP	XCUP
CWD	XCWD

All FTP implementations **should** recognize both forms of these commands, by simply equating them with extra entries in the command lookup table.

Implementation

A User-FTP can access a server that supports only the "X" forms by implementing a mode switch, or automatically using the following procedure: if the RFC-959 form of one of the above commands is rejected with a 500 or 502 response code, then try the experimental form; any other response would be passed to the user.

Idle Timeout

A Server-FTP process **should** have an idle timeout, which will terminate the process and close the control connection if the server is inactive (i.e., no commandor data transfer in progress) for a long period of time. The idle timeout time **should** be configurable, and the default should be at least 5 minutes.

A client FTP process ("User-PI" in RFC-959) will need timeouts on responses only if it is invoked from a program.

Discussion

Without a timeout, a Server-FTP process may be left pending indefinitely if the corresponding client crashes without closing the control connection.

Concurrency of Data and Control

Discussion

The intent of the designers of FTP was that a user should be able to send a STAT command at any time while data transfer was in progress and that the server-FTP would reply immediately with status -- e.g., the number of bytes transferred so far. Similarly, an ABOR command should be possible at any time during a data transfer.

Unfortunately, some small-machine operating systems make such concurrent programming difficult, and some other implementers seek minimal solutions, so some FTPimplementations do not allow concurrent use of the data and control connections. Even such a minimal server must be prepared to accept and defer a STAT or ABOR command that arrives during data transfer.

User Interface

This section discusses the user interface for a User-FTP program.

Pathname Specification

Since FTP is intended for use in a heterogeneous environment, User-FTP implementations **must** support remote pathnames as arbitrary character strings, so that their form and content are not limited by the conventions of the local operating system.

Discussion

In particular, remote pathnames can be of arbitrary length, and all the printing ASCII characters as well as space (0x20) must be allowed. RFC-959allows a pathname to contain any 7-bit ASCII character except CR or LF.

Displaying Replies to User

A User-FTP **should** display to the user the full text of all error reply messages it receives. It **should** have a "verbose" mode in which all commands it sends and the full text and reply codes it receives are displayed, for diagnosis of problems.

Maintaining Synchronization

The state machine in a User-FTP **should** be forgiving of missing and unexpected reply messages, in order to maintain command synchronization with the server.

Application Layer General Issues

This section contains general requirements that may be applicable to all application-layer protocols.

General Issues

Host Names and Numbers

The syntax of a legal Internet host name was specified in "DoD Internet Host Table Specification" [RFC-952]. One aspect of host name syntax is hereby changed: the restriction on the first character is relaxed to allow either a letter or a digit. Host software **must** support this more liberal syntax.

Host software **must** handle host names of up to 63 characters and **should** handle host names of up to 255 characters.

Whenever a user inputs the identity of an Internet host, it **should** be possible to enter either (1) a host domain name or (2) an IP address in dotted-decimal ("#.#.#.#") form. The host **should** check the string syntactically for a dotted-decimal number before looking it up in the Domain Name System.

Disscussion

This last requirement is not intended to specify the complete syntactic form for entering a dotted-decimal host number; that is considered to be a user-interface issue. For example, a dotted-decimal number **must** be enclosed within "[ ]" brackets for SMTP mail. This notation could be made universal within a host system, simplifying the syntactic checking for a dotted-decimal number.

If a dotted-decimal number can be entered without such identifying delimiters, then a full syntactic check **must** be made, because a segment of a host domain name is now allowed to begin with a digit and could legally be entirely numeric. However, a valid host name can never have the dotted-decimal form #.#.#.#, since at least the highest-level component label will be alphabetic.

Using Domain Name Service

Host domain names **must** be translated to IP addresses as described in Domain Name Translation.

Applications using domain name services **must** be able to cope with soft error conditions. Applications **must** wait a reasonable interval between successive retries due to a soft error, and **must** allow for the possibility that network problems may deny service for hours or even days.

An application **should not** rely on the ability to locate a WKS record containing an accurate listing of all services at a particular host address, since the WKSRR type is not often used by Internet sites. To confirm that a service is present, simply attempt to use it.

Applications on Multihomed hosts

When the remote host is multihomed, the name-to-address translation will return a list of alternative IP addresses. This listshould be in order of decreasing preference. Application protocol implementations **should** be prepared to try multiple addresses from the list until success is obtained. More specific requirements for SMTP are given in SMTP Overview.

When the local host is multihomed, a UDP-based request/response application **should** send the response with an IP source address that is the same as the specific destination address of the UDP request datagram. The "specific destination address" is defined in the "Additional Addressing Notes" section of RFC-791 [RFC-791].

Similarly, a server application that opens multiple TCP connections to the same client **should** use the same local IP address for all.

General Issues

Type-of-Service Issues for Applications

Applications **must** select appropriate TOS values when they invoke transport layer services, and these values **must** be configurable. Note that a TOS value contains 5 bits, of which only the most- significant 3 bits are currently defined; the other two bits **must** be zero.

Discussion

As gateway algorithms are developed to implement Type-of- Service, the recommended values for various application protocols may change. In addition, it is likely that particular combinations of users and Internet paths will want non-standard TOS values. For these reasons, the TOS values **must** be configurable.

See the latest version of the "Assigned Numbers" RFC [RFC-1060] for the recommended TOS values for the major application protocols.

Host Initialization

Host Initialization

Introduction

This section discusses the initialization of host software across a connected network, or more generally across an Internet path. This is necessary for a diskless host, and may optionally be used for a host with disk drives. For a diskless host, the initialization process is called "network booting" and is controlled by a bootstrap program located in a boot ROM.

To initialize a diskless host across the network, there are two distinct phases:

(1) Configure the IP layer.

Diskless machines often have no permanent storage in which to store network configuration information, so that sufficient configuration information must be obtained dynamically to support the loading phase that follows. This information must include at least the IP addresses of the host and of the boot server. To support booting across a gateway, the address mask and a list of default gateways are also required.

(2) Load the host system code.

During the loading phase, an appropriate file transfer protocol is used to copy the system code across the network from the boot server.

A host with a disk may perform the first step, dynamic configuration. This is important for microcomputers, whose floppy disks allow network configuration information to be mistakenly duplicated on more than one host. Also, installation of new hosts is much simpler if they automatically obtain their configuration information from a central server, saving administrator time and decreasing the probability of mistakes.

Dynamic Configuration

A number of protocol provisions have been made for dynamic configuration.

- ICMP Information Request/Reply messages

  This **obsolete** message pair was designed to allow a host to find the number of the network it is on. Unfortunately, it was useful only if the host already knewthe host number part of its IP address, information that hosts requiring dynamic configuration seldom had.

- Reverse Address Resolution Protocol (RARP)

  RARP is a link-layer protocol for a broadcast medium that allows a host to find its IP address given its link layer address. Unfortunately, RARP does not workacross IP gateways and therefore requires a RARP server on every network. In addition, RARP does not provide any other configuration information.

- ICMP Address Mask Request/Reply messages

  These ICMP messages allow a host to learn the address mask for a particular network interface.

- BOOTP Protocol

  This protocol allows a host to determine the IP addresses of the local host and the boot server, the name of an appropriate boot file, and optionally the address mask and list of default gateways. To locate a BOOTP server, the host broadcasts a BOOTP request using UDP. Ad hoc gateway extensions have been used to transmit the BOOTP broadcast through gateways, and in the future the IP Multicasting facility will provide a standard mechanism for this purpose.

The suggested approach to dynamic configuration is to use the BOOTP protocol with the extensions defined in "BOOTP Vendor Information Extensions" RFC-1084. RFC-1084 defines some important general (not vendor-specific) extensions. In particular, these extensions allow the address mask to be supplied in BOOTP; we **recommend** that the address mask be supplied in this manner.

Discussion

Historically, subnetting was defined long after IP, and so a separate mechanism (ICMP Address Mask messages) was designed to supply the address mask to a host. However, the IP address mask and the corresponding IP address conceptually form a pair, and for operational simplicity they ought to be defined at the same time and by the same mechanism, whether a configuration file or a dynamic mechanism like BOOTP.

Note that BOOTP is not sufficiently general to specify the configurations of all interfaces of a multihomed host. A multihomed host must either use BOOTP separately for each interface, or configure one interface using BOOTP to perform the loading, and perform the complete initialization from a file later.

Application layer configuration information is expected to be obtained from files after loading of the system code.

Loading Phase

A suggested approach for the loading phase is to use TFTP [RFC-906] between the IP addresses established by BOOTP [RFC-951].

TFTP to a broadcast address **should not** be used.

Remote Management

Introduction

The Internet community has recently put considerable effort into the development of network management protocols. The result has been a two-pronged approach [RFC-1052, RFC-1109]: the Simple Network Management Protocol (SNMP) [RFC-1157] and the Common Management Information Protocol over TCP (CMOT) [RFC-1189].

In order to be managed using SNMP or CMOT, a host will need to implement an appropriate management agent. An Internet host **should** include an agent for either SNMP or CMOT.

Both SNMP and CMOT operate on a Management Information Base (MIB) that defines a collection of management values. By reading and setting these values, a remote application may query and change the state of the managed system.

A standard MIB [RFC-1156] has been defined for use by both management protocols, using data types defined by the Structure of Management Information (SMI) defined in [RFC-1155]. Additional MIB variables can be introduced under the "enterprises" and "experimental" subtrees of the MIB naming space [RFC-1155].

Every protocol module in the host **should** implement the relevant MIB variables. A host **should** implement the MIB variables as defined in the most recent standard MIB, and **may** implement other MIB variables when appropriate and useful.

Protocol Walk-Through

The MIB is intended to cover both hosts and gateways, although there may be detailed differences in MIB application to the two cases. This section contains the appropriate interpretation of the MIB for hosts. It is likely that later versions of the MIB will include more entries for host management.

A managed host must implement the following groups of MIB object definitions: System, Interfaces, Address Translation, IP, ICMP, TCP, and UDP.

The following specific interpretations apply to hosts:

- **ipInHdrErrors**

  Note that the error "time-to-live exceeded" can occur in a host only when it is forwarding a source-routed datagram.

- **ipOutNoRoutes**

  This object counts datagrams discarded because no route can be found. This may happen in a host if all the default gateways in the host's configuration are down.

- **ipFragOKs, ipFragFails, ipFragCreates**

  A host that does not implement intentional fragmentation (see "Fragmentation" section of [INTRO:1]) MUST return the value zero for these three objects.

- **icmpOutRedirects**

  For a host, this object MUST always be zero, since hosts do not send Redirects.

- **icmpOutAddrMaskReps**

  For a host, this object MUST always be zero, unless the host is an authoritative source of address mask information.

- **ipAddrTable**

  For a host, the "IP Address Table" object is effectively a table of logical interfaces.

- **ipRoutingTable**

  For a host, the "IP Routing Table" object is effectively a combination of the host's Routing Cache and the static route table described in Routing Outbound Datagrams.

  Within each ipRouteEntry, ipRouteMetric1...4 normally will have no meaning for a host and **should** always be -1, while ipRouteType will normally have the value "remote".

If destinations on the connected network do not appear in the Route Cache, there will be no entries withipRouteType of "direct".

Discussion

The current MIB does not include Type-of-Service in an ipRouteEntry, but a future revision is expected to make this addition.

We also expect the MIB to be expanded to allow the remote management of applications (e.g., the ability to partially reconfigure mail systems). Network service applications such as mail systems should therefore be written with the "hooks" for remote management.

Electronic Mail -- SMTP and RFC-822

Introduction

In the TCP/IP protocol suite, electronic mail in a format specified in RFC-822 is transmitted using the Simple Mail Transfer Protocol (SMTP) defined in RFC-821.

While SMTP has remained unchanged over the years, the Internet community has made several changes in the way SMTP is used. In particular, the conversion to the Domain Name System (DNS) has caused changes in address formats and in mail routing. In this section, we assume familiarity with the concepts and terminologyof the DNS, whose requirements are given in Domain Name Service.

RFC-822 entitled Standard for the Format of ARPA Internet Text Messagessupercedes an older standard, RFC-733, that may still be in use in a few places, although it is obsolete. The two formats are sometimes referred to simply by number ("822" and "733").

RFC-822 is used in some non-Internet mail environments with different mail transfer protocols than SMTP, and SMTP has also been adapted for use in some non-Internet environments. Note that this document presents the rules for the use of SMTP and RFC-822 for the Internet environment only; other mail environments thatuse these protocols may be expected to have their own rules.

Specific Issues

Specific Issues

Queueing Strategies

The common structure of a host SMTP implementation includes user mailboxes, one or more areas for queueing messages in transit, and one or more daemon processes for sending and receiving mail. The exact structure will vary depending on the needs of the users on the host and the number and size of mailing lists supported by the host. We describe several optimizations that have proved helpful, particularly for mailers supporting high traffic levels.

Any queueing strategy **must** include:

- Timeouts on all activities.

- Never sending error messages in response to error messages.

Sending Strategy

The general model of a sender-SMTP is one or more processes that periodically attempt to transmit outgoing mail. In a typical system, the program that composes a message has some method for requesting immediate attention for a new piece of outgoing mail, while mail that cannot be transmitted immediately **must** be queued and periodically retried by the sender. A mail queue entry will include not only the message itself but also the envelope information.

The sender **must** delay retrying a particular destination after one attempt has failed. In general, the retry interval **should** be at least 30 minutes; however, more sophisticated and variable strategies will be beneficial when the sender-SMTP can determine the reason for non- delivery.

Retries continue until the message is transmitted or the sender gives up; the give-up time generally needs to be at least 4-5 days. The parameters to the retry algorithm **must** be configurable.

A sender **should** keep a list of hosts it cannot reach and corresponding timeouts, rather than just retrying queued mail items.

> **Discussion**
>
> Experience suggests that failures are typically transient (the target system has crashed), favoring a policy of two connection attempts in the first hour the message is in the queue, and then backing off to once every two or three hours.
>
> The sender-SMTP can shorten the queueing delay by cooperation with the receiver-SMTP. In particular, if mail is received from a particular address, it is goodevidence that any mail queued for that host can now be sent.
>
> The strategy may be further modified as a result of multiple addresses per host, to optimize delivery time vs. resource usage.
>
> A sender-SMTP may have a large queue of messages for each unavailable destination host, and if it retried all these messages in every retry cycle, there would be excessive Internet overhead and the daemon would be blocked for a long period. Note that an SMTP can generally determine that a delivery attempt has failedonly after a timeout of a minute or more; a one minute timeout per connection will result in a very large delay if it is repeated for dozens or even hundreds of queued messages.

When the same message is to be delivered to several users on the same host, only one copy of the message **should** be transmitted. That is, the sender-SMTP should use the command sequence: RCPT, RCPT,... RCPT, DATA instead of the sequence: RCPT, DATA, RCPT, DATA,... RCPT, DATA. Implementation of this efficiency feature is strongly urged.

Similarly, the sender-SMTP **may** support multiple concurrent outgoing mail transactions to

achieve timely delivery. However, some limit **should** be imposed to protect the host from devoting all its resources to mail.

Receiving Strategy

The receiver-SMTP **should** attempt to keep a pending listen on the SMTP port at all times. This will require the support of multiple incoming TCP connections for SMTP. Some limit **may** be imposed.

Implementation

When the receiver-SMTP receives mail from a particular host address, it could notify the sender-SMTP to retry any mail pending for that host address.

Timeouts

There are two approaches to timeouts in the sender-SMTP: (a) limit the time for each SMTP command separately, or (b) limit the time for the entire SMTP dialogue for a single mail message. A sender-SMTP **should** use option (a), per-command timeouts. Timeouts **should** be easily reconfigurable, preferably without recompiling the SMTP code.

Discussion

Timeouts are an essential feature of an SMTP implementation. If the timeouts are too long (or worse, there are no timeouts), Internet communication failures or software bugs in receiver-SMTP programs can tie up SMTP processes indefinitely. If the timeouts are too short, resources will be wasted with attempts that time out part way through message delivery.

If option (b) is used, the timeout has to be very large, e.g., an hour, to allow time to expand very large mailing lists. The timeout may also need to increase linearly with the size of the message, to account for the time to transmit a very large message. A large fixed timeout leads to two problems: a failure can still tie up the sender for a very long time, and very large messages may still spuriously time out (which is a wasteful failure!).

Using the recommended option (a), a timer is set for each SMTP command and for each buffer of the data transfer. The latter means that the overall timeout is inherently proportional to the size of the message.

Per Command Timeouts

Based on extensive experience with busy mail-relay hosts, the minimum per-command timeout values **should** be as follows:

- Initial 220 Message: 5 minutes

  A Sender-SMTP process needs to distinguish between a failed TCP connection and a delay in receiving the initial 220 greeting message. Many receiver-SMTPs willaccept a TCP connection but delay delivery of the 220 message until their system load will permit more mail to be processed.

- MAIL Command: 5 minutes

- RCPT Command: 5 minutes

  A longer timeout would be required if processing of mailing lists and aliases were not deferred until after the message was accepted.

- DATA Initiation: 2 minutes

  This is while awaiting the "354 Start Input" reply to a DATA command.

- Data Block: 3 minutes

  This is while awaiting the completion of each TCP SEND call transmitting a chunk of data.

- DATA Termination: 10 minutes.

  This is while awaiting the "250 OK" reply. When the receiver gets the final period terminating the message data, it typically performs processing to deliver the message to a user mailbox. A spurious timeout at this point would be very wasteful, since the message has been successfully sent.

A receiver-SMTP **should** have a timeout of at least 5 minutes while it is awaiting the next command from the sender.

Reliable Mail Receipt

When the receiver-SMTP accepts a piece of mail (by sending a "250 OK" message in response to DATA), it is accepting responsibility for delivering or relaying the message. It must take this responsibility seriously, i.e., it **must not** lose the message for frivolous reasons, e.g., because the host later crashes or because of a predictable resource shortage.

If there is a delivery failure after acceptance of a message, the receiver-SMTP **must** formulate and mail a notification message. This notification **must** be sentusing a null ("<>") reverse path in the envelope; see Section 3.6 of RFC-821. The recipient of this notification **should** be the address from the envelope return path (or the Return-Path: line). However, if this address is null ("<>"), the receiver-SMTP **must not** send a notification. If the address is an explicit source route, it **should** be stripped down to its final hop.

> **Discussion**
>
> For example, suppose that an error notification must be sent for a message that arrived with: "MAIL FROM:<@a,@b:user@d>". The notification messageshould be sent to: "RCPT TO:<user@d>".
>
> Some delivery failures after the message is accepted by SMTP will be unavoidable. For example, it may be impossible for the receiver-SMTP to validate all the delivery addresses in RCPT command(s) due to a "soft" domain system error or because the target is a mailing list (see earlier discussion of RCPT).

To avoid receiving duplicate messages as the result of timeouts, a receiver-SMTP **must** seek to minimize the time required to respond to the final "." that ends a message transfer. See RFC-1047 [SMTP:4] for a discussion of this problem.

Simple Mail Transfer Protocol

Reliable Mail Transmission

To transmit a message, a sender-SMTP determines the IP address of the target host from the destination address in the envelope. Specifically, it maps the string to the right of the "@" sign into an IP address. This mapping or the transfer itself may fail with a soft error, in which case the sender- SMTP will requeue the outgoing mail for a later retry.

When it succeeds, the mapping can result in a list of alternative delivery addresses rather than a single address, because of (a) multiple MX records, (b) multihoming, or both. To provide reliable mail transmission, the sender-SMTP **must** be able to try (and retry) each of the addresses in this list in order, until a delivery attempt succeeds. However, there **may** also be a configurable limit on the number of alternate addresses that can be tried. In any case, a host **should** try at least two addresses.

The following information is to be used to rank the host addresses:

(1) Multiple MX Records -- these contain a preference indication that should be used in sorting. If there are multiple destinations with the same preference and there is no clear reason to favor one (e.g., by address preference), then the sender-SMTP **should** pick one at random to spread the load across multiple mail exchanges for a specific organization; note that this is a refinement of the procedure in [DNS:3].

(2) Multihomed host -- The destination host (perhaps taken from the preferred MX record) may be multihomed, in which case the domain name resolver will return a list of alternative IP addresses. It is the responsibility of the domain name resolver interface (see Section 6.1.3.4 below) to have ordered this list by decreasing preference, and SMTP **must** try them in the order presented.

Discussion

Although the capability to try multiple alternative addresses is required, there may be circumstances where specific installations want to limit ordisable the use of alternative addresses. The question of whether a sender should attempt retries using the different addresses of a multihomed host has been controversial. The main argument for using the multiple addresses is that it maximizes the probability of timely delivery, and indeed sometimes the probability of any delivery; the counter argument is that it may result in unnecessary resource use.

Note that resource use is also strongly determined by the sending strategy.

Domain Name Support

SMTP implementations **must** use the mechanism defined in Section 6.1 for mapping between domain names and IP addresses. This means that every Internet SMTP **must** include support for the Internet DNS.

In particular, a sender-SMTP **must** support the MX record scheme [SMTP:3]. See also Section 7.4 of [DNS:2] for information on domain name support for SMTP.

Mailing Lists and Aliases

An SMTP-capable host **should** support both the alias and the list form of address expansion for multiple delivery. When a message is delivered or forwarded to each address of an expanded list form, the return address in the envelope ("MAIL FROM:") **must** be changed to be the address of a person who administers the list,but the message header **must** be left unchanged; in particular, the "From" field of the message is unaffected.

Discussion

An important mail facility is a mechanism for multi- destination delivery of a single message, by transforming or "expanding" a pseudo-mailbox address into a list of destination mailbox addresses. When a message is sent to such a pseudo-mailbox (sometimes called an "exploder"), copies are forwarded or redistributed to each mailbox in the expanded list. We classify such a pseudo-mailbox as an "alias" or a "list", depending upon the expansion rules:

(a) **Alias**

To expand an alias, the recipient mailer simply replaces the pseudo-mailbox address in the envelope with each of the expanded addresses in turn; the rest of the envelope and the message body are left unchanged. The message is then delivered or forwarded to each expanded address.

(b) **List**

A mailing list may be said to operate by "redistribution" rather than by "forwarding". To expand a list, the recipient mailer replaces the pseudo-mailbox address in the envelope with each of the expanded addresses in turn. The return address in the envelope is changed so that all error messages generated by the final deliveries will be returned to a list administrator, not to the message originator, who generally has no control over the contents of the list and will typically find error messages annoying.

Mail Gatewaying

Gatewaying mail between different mail environments, i.e., different mail formats and protocols, is complex and does not easily yield to standardization. See for example [SMTP:5a], [SMTP:5b]. However, some general requirements may be given for a gateway between the Internet and another mail environment.

(A) Header fields **may** be rewritten when necessary as messages are gatewayed across mail environment boundaries.

> **Discussion**
> This may involve interpreting the local-part of the destination address.
>
> The other mail systems gatewayed to the Internet generally use a subset of RFC-822 headers, but some of them do not have an equivalent to the SMTP envelope. Therefore, when a message leaves the Internet environment, it may be necessary to fold the SMTP envelope information into the message header. A possible solution would be to create new header fields to carry the envelope information (e.g., "X- SMTP-MAIL:" and "X-SMTP-RCPT:"); however, this would require changes in mail programs in the foreign environment.

(B) When forwarding a message into or out of the Internet environment, a gateway **must** prepend a Received: line, but it **must not** alter in any way a Received: line that is already in the header.

> **Discussion**
> This requirement is a subset of the general "Received:" line requirement in DATA Command; it is restated here for emphasis.
>
> Received: fields of messages originating from other environments may not conform exactly to RFC-822most important use of Received: lines is for debugging mail faults, and this debugging can be severely hampered by well-meaning gateways that try to "fix" a Received: line.
>
> The gateway is strongly encouraged to indicate the environment and protocol in the "via" clauses of Received field(s) that it supplies.

(C) From the Internet side, the gateway **should** accept all valid address formats in SMTP commands and in RFC-822 headers, and all valid RFC-822 messages. Although a gateway must accept an RFC-822 explicit source route ("@...:" format) in either the RFC-822 header or in the envelope, it **may** or may not act on the source route; see Mail Relay and Explicit Source Route.

> **Discussion**
> It is often tempting to restrict the range of addresses accepted at the mail gateway to simplify the translation into addresses for the remote environment. This practice is based on the

assumption that mail users have control over the addresses their mailers send to the mail gateway. In practice, however, users have little control over the addresses that are finally sent; their mailers are free to change addresses into any legal RFC-822 format.

(D) The gateway **must** ensure that all header fields of a message that it forwards into the Internet meet the requirements for Internet mail. In particular, alladdresses in "From:", "To:", "Cc:", etc., fields must be transformed (if necessary) to satisfy RFC-822 syntax, and they must be effective and useful for sending replies.

(E) The translation algorithm used to convert mail from the Internet protocols to another environment's protocol **should** try to ensure that error messages from the foreign mail environment are delivered to the return path from the SMTP envelope, not to the sender listed in the "From:" field of the RFC-822 message.

Discussion
Internet mail lists usually place the address of the mail list maintainer in the envelope but leave the original message header intact (with the "From:" field containing the original sender). This yields the behavior the average recipient expects: a reply to the header gets sent to the original sender, not to a mail list maintainer; however, errors get sent to the maintainer (who can fix the problem) and not the sender (who probably cannot).

(F) Similarly, when forwarding a message from another environment into the Internet, the gateway **should** set the envelope return path in accordance with an error message return address, if any, supplied by the foreign environment.

Maximum Message Size

Mailer software **must** be able to send and receive messages of at least 64K bytes in length (including header), and a much larger maximum size is highly desirable.

Discussion

Although SMTP does not define the maximum size of a message, many systems impose implementation limits.

For example, mail is often used instead of FTP for transmitting ASCII files, and in particular to transmit entire documents. As a result, messages can be 1 megabyte or even larger.

Trivial File Transfer Protocol
TFTP

Introduction

The Trivial File Transfer Protocol TFTP is defined in RFC-783.

TFTP provides its own reliable delivery with UDP as its transport protocol, using a simple stop-and-wait acknowledgment system. Since TFTP has an effective window of only one 512 octet segment, it can provide good performance only over paths that have a small delay*bandwidth product. The TFTP file interface is very simple, providing no access control or security.

TFTP's most important application is bootstrapping a host over a local network, since it is simple and small enough to be easily implemented in EPROM. Vendors are urged to support TFTP for booting.

Specific Issues

Specific Issues

Sorcerer's Apprentice Syndrome

There is a serious bug, known as the "Sorcerer's Apprentice Syndrome," in the protocol specification. While it does not cause incorrect operation of the transfer (the file will always be transferred correctly if the transfer completes), this bug may cause excessive retransmission, which may cause the transfer to time out.

Implementations **must** contain the fix for this problem: the sender (i.e., the side originating the DATA packets) must never resend the current DATA packet on receipt of a duplicate ACK.

Discussion

The bug is caused by the protocol rule that either side, on receiving an old duplicate datagram, may resend the current datagram. If a packet is delayed in the network but later successfully delivered after either side has timed out and retransmitted a packet, a duplicate copy of the response may be generated. If the other side responds to this duplicate with a duplicate of its own, then every datagram will be sent in duplicate for the remainder of the transfer (unless a datagram is lost, breaking the repetition). Worse yet, since the delay is often caused by congestion, this duplicate transmission will usually causes more congestion, leading to more delayed packets, etc. (See Example).

Trivial File Transfer Protocol

Sorcerer's Apprentice Syndrome Example

	TFTP A	TFTP B
(1)	Receive ACK X-1 Send DATA X	
(2)		Receive DATA X Send ACK X

(ACK X is delayed in network, and A times out):

(3)	Retransmit DATA X	
(4)		Receive DATA X again Send ACK X again
(5)	Receive (delayed) ACK X Send DATA X+1	
(6)		Receive DATA X+1 Send ACK X+1
(7)	Receive ACK X again Send DATA X+1 again	
(8)		Receive DATA X+1 again Send ACK X+1 again
(9)	Receive ACK X+1 Send DATA X+2	
(10)		Receive DATA X+2 Send ACK X+3
(11)	Receive ACK X+1 again Send DATA X+2 again	
(12)		Receive DATA X+2 again Send ACK X+3 again

Notice that once the delayed ACK arrives, the protocol settles down to duplicate all further packets (sequences 5-8 and 9-12). The problem is caused not by either side timing out, but by both sides retransmitting the current packet when they receive a duplicate.

The fix is to break the retransmission loop, as indicated above. This is analogous to the behavior of TCP. It is then possible to remove the retransmission timer on the receiver, since the resent ACK will never cause any action; this is a useful simplification where TFTP is used in a bootstrap program. It is OK toallow the timer to remain, and it may be helpful if the retransmitted ACK replaces one that was genuinely lost in the network. The sender still requires a retransmit timer, of course.

Timeout Algorithms

A TFTP implementation **must** use an adaptive timeout.

Implementation

TCP retransmission algorithms provide a useful base to work from. At least an exponential backoff of retransmission timeout is necessary.

Extensions

A variety of non-standard extensions have been made to TFTP, including additional transfer modes and a secure operation mode (with passwords). None of these have been standardized.

Access Control

A server TFTP implementation **should** include some configurable access control over what pathnames are allowed in TFTP operations.

Broadcast Request

A TFTP request directed to a broadcast address **should** be silently ignored.

Discussion

Due to the weak access control capability of TFTP, directed broadcasts of TFTP requests to random networks could create a significant security hole.

Remote Login -- TELNET Protocol

Telnet is the standard Internet application protocol for remote login. It provides the encoding rules to link a user's keyboard/display on a client ("user") system with a command interpreter on a remote server system. A subset of the Telnet protocol is also incorporated within other application protocols, e.g., FTP and SMTP.

Telnet uses a single TCP connection, and its normal data stream ("Network Virtual Terminal" or "NVT" mode) is 7-bit ASCII with escape sequences to embed control functions. Telnet also allows the negotiation of many optional modes and functions.

The primary Telnet specification is to be found in "RFC-854 Telnet Protocol Specification" [RFC-854], while the options are defined in many other RFCs; see References.

Internet Group Management Protocol (IGMP)

This specification originally appeared as an appendix to RFC-1112, Host Extensions for IP Multicasting.

IGMP is a protocol used between hosts and gateways on a single network to establish hosts' membership in particular multicast groups. The gateways use this information, in conjunction with a multicast routing protocol, to support IP multicasting across the Internet.

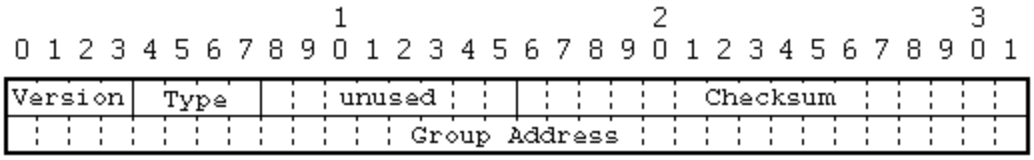
At this time, implementation of IGMP is **optional**; see IP Multicasting for more information. Without IGMP, a host can still participate in multicasting local to its connected networks.

Internet Group Management Protocol

Message Format

The Internet Group Management Protocol (IGMP) is used by IP hosts to report their host group memberships to any immediately-neighboring multicast routers. IGMP is an asymmetric protocol and is specified here from the point of view of a host, rather than a multicast router. (IGMP may also be used, symmetrically or asymmetrically, between multicast routers. Such use is not specified here.)

Like ICMP, IGMP is a integral part of IP. It is required to be implemented by all hosts conforming to level 2 of the IP multicasting specification. IGMP messages are encapsulated in IP datagrams, with an IP protocol number of 2. All IGMP messages of concern to hosts have the following format:



Version

This memo specifies version 1 of IGMP. Version 0 is specified in RFC-988 and is now obsolete.

Type

There are two types of IGMP message of concern to hosts:

1 = Host Membership Query

2 = Host Membership Report

Unused

Unused field, zeroed when sent, ignored when received.

Checksum

Group Address

In a Host Membership Query message, the group address field is zeroed when sent, ignored when received.

In a Host Membership Report message, the group address field holds the IP host group address of the group being reported.

Internet Group Management Protocol

Informal Protocol Description

Multicast routers send Host Membership Query messages (hereinafter called Queries) to discover which host groups have members on their attached local networks. Queries are addressed to the all-hosts group (address 224.0.0.1), and carry an IP time-to-live of 1.

Hosts respond to a Query by generating Host Membership Reports (hereinafter called Reports), reporting each host group to which they belong on the network interface from which the Query was received. In order to avoid an "implosion" of concurrent Reports and to reduce the total number of Reports transmitted, two techniques are used:

1. When a host receives a Query, rather than sending Reports immediately, it starts a report delay timer for each of its group memberships on the network interface of the incoming Query. Each timer is set to a different, randomly-chosen value between zero and D seconds. When a timer expires, a Report is generated for the corresponding host group. Thus, Reports are spread out over a D second interval instead of all occurring at once.

2. A Report is sent with an IP destination address equal to the host group address being reported, and with an IP time-to-live of 1, so that other members of the same group on the same network can overhear the Report. If a host hears a Report for a group to which it belongs on that network, the host stops its own timer for that group and does not generate a Report for that group. Thus, in the normal case, only one Report will be generated for each group present on the network, by the member host whose delay timer expires first. Note that the multicast routers receive all IP multicast datagrams, and therefore need not be addressed explicitly. Further note that the routers need not know which hosts belong to a group, only that at least one host belongs to a group on a particular network.

There are two exceptions to the behavior described above. First, if a report delay timer is already running for a group membership when a Query is received, that timer is not reset to a new random value, but rather allowed to continue running with its current value. Second, a report delay timer is never set for a host's membership in the all- hosts group (224.0.0.1), and that membership is never reported.

If a host uses a pseudo-random number generator to compute the reporting delays, one of the host's own individual IP address should be used as part of the seed for the generator, to reduce the chance of multiple hosts generating the same sequence of delays.

A host should confirm that a received Report has the same IP host group address in its IP destination field and its IGMP group address field, to ensure that the host's own Report is not cancelled by an erroneous received Report. A host should quietly discard any IGMP message of type other than Host Membership Query or Host Membership Report.

Multicast routers send Queries periodically to refresh their knowledge of memberships present on a particular network. If no Reports are received for a particular group after some number of Queries, the routers assume that that group has no local members and that they need not forward remotely-originated multicasts for that group onto the local network. Queries are normally sent infrequently (no more than once a minute) so as to keep the IGMP overhead on hosts and networks very low. However, when a multicast router starts up, it may issue several closely-spaced Queries in order to build up its knowledge of local

memberships quickly.

When a host joins a new group, it should immediately transmit a Report for that group, rather than waiting for a Query, in case it is the first member of that group on the network. To cover the possibility of the initial Report being lost or damaged, it is recommended that it be repeated once or twice after short delays. (A simple way to accomplish this is to act as if a Query had been received for that group only, setting the group's random report delay timer. The state transition diagram below illustrates this approach.)

Note that, on a network with no multicast routers present, the only IGMP traffic is the one or more Reports sent whenever a host joins a new group.

State Transition Diagram

IGMP behavior is more formally specified by the state transition diagram below. A host may be in one of three possible states, with respect to any single IP host group on any single network interface:

- Non-Member state, when the host does not belong to the group on the interface. This is the initial state for all memberships on all network interfaces; it requires no storage in the host.
- Delaying Member state, when the host belongs to the group on the interface and has a report delay timer running for that membership.
- Idle Member state, when the host belongs to the group on the interface and does not have a report delay timer running for that membership.

There are five significant events that can cause IGMP state transitions:

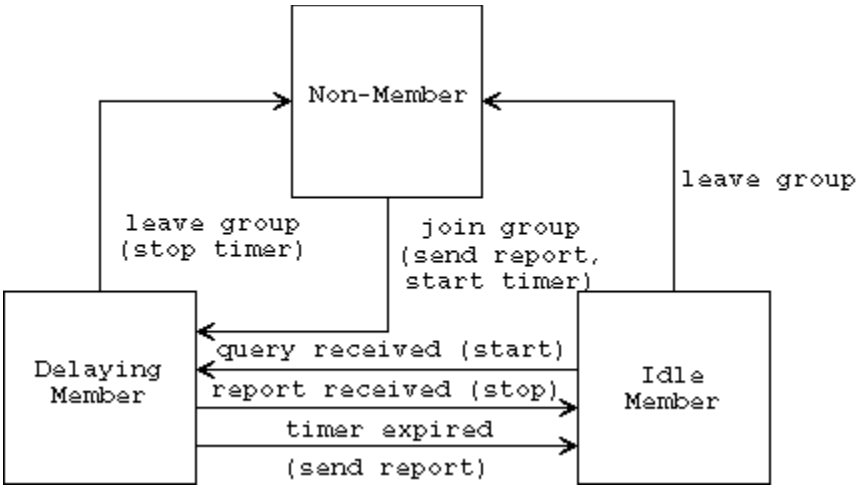
- "join group" occurs when the host decides to join the group on the interface. It may occur only in the Non-Member state.
- "leave group" occurs when the host decides to leave the group on the interface. It may occur only in the Delaying Member and Idle Member states.
- "query received" occurs when the host receives a valid IGMP Host Membership Query message. To be valid, the Query message must be at least 8 octets long, have a correct IGMP checksum and have an IP destination address of 224.0.0.1. A single Query applies to all memberships on the interface from which the Query is received. It is ignored for memberships in the Non-Member or Delaying Member state.
- "report received" occurs when the host receives a valid IGMP Host Membership Report message. To be valid, the Report message must be at least 8 octets long, have a correct IGMP checksum, and contain the same IP host group address in its IP destination field and its IGMP group address field. A Report applies only to the membership in the group identified by the Report, on the interface from which the Report is received. It is ignored for memberships in the Non-Member or Idle Member state.
- "timer expired" occurs when the report delay timer for the group on the interface expires. It may occur only in the Delaying Member state.

All other events, such as receiving invalid IGMP messages, or IGMP messages other than Query or Report, are ignored in all states.

There are three possible actions that may be taken in response to the above events:

- "send report" for the group on the interface.
- "start timer" for the group on the interface, using a random delay value between 0 and D seconds.
- "stop timer" for the group on the interface.

In the following diagram, each state transition arc is labelled with the event that causes the transition, and, in parentheses, any actions taken during the transition.



The all-hosts group (address 224.0.0.1) is handled as a special case. The host starts in Idle Member state for that group on every interface, never transitions to another state, and never sends a report for that group.

The maximum report delay, D, is 10 seconds.

Numerical Index of RFC's

RFC-952	**DoD Internet Host Table Specification**
RFC-953	**Hostname Server Protocol**
RFC-954	**NicName/Whois Protocol**
RFC-959	**The File Transfer Protocol (FTP)**
RFC-974	**Mail Routing and the Domain System**
RFC-1001	**Protocol Standard for a NetBIOS Service: Concepts and Methods**
RFC-1002	**Protocol Standard for a NetBIOS Service: Detailed Specifications**
RFC-1006	**ISO Transport Services on top of the TCP: Version 3**
RFC-1014	**XDR: External Data Representation Standard**
RFC-1032	**Domain Administrators Guide**
RFC-1033	**Domain Administrators Operations Guide**
RFC-1034	**Domain Names - Concepts and Facilities**
RFC-1035	**Domain Names - Implementation and Specification**
RFC-1042	**A Standard for the Transmission of IP Datagrams on IEEE 802 Networks**
RFC-1049	**Content-Type Header Field for Internet Mail**
RFC-1055	**A Non-Standard for the Transmission of IP Datagrams over Serial Lines**
RFC-1057	**RPC: Remote Procedure Protocol**
RFC-1060	**Assigned Numbers from March 1990**
RFC-1084	**BOOTP Vendor Information Extensions**
RFC-1088	**A Standard for the Transmission of IP Datagrams over NetBIOS Networks**
RFC-1091	**Telnet Terminal-Type Option**
RFC-1094	**NFS: Network File System Protocol Specificiation**
RFC-1101	**DNS Encoding of Network Names and Other Types**
RFC-1112	**Host Extensions for IP Multicasting (including IGMP)**
RFC-1122	**Requirements for Internet Hosts - Communications Layer** (Incorporated into appropriate RFCs)
RFC-1123	**Requirements for Internet Hosts - Application and Support** (Incorporated into appropriate RFCs)
RFC-1147	**FYI on Network Monitoring and Management Tools**
RFC-1155	**Structure and Identification of Management Information for TCP/IP-based Internets**
RFC-1156	**Management Information Base for the Management of TCP/IP-Based Internets**
RFC-1157	**A Simple Network Management Protocol (SNMP)**
RFC-1160	**The Internet Activities Board**
RFC-1171	**Point-to-Point Protocol for the Transmission of Multi-Protocol Datagrams over Point-to-Point Links**
RFC-1189	**Common Management Information Services and Protocols for the Internet (CMOT)**
RFC-1183	**New DNS RR Definitions**
RFC-1196	**Finger User Information Protocol**
RFC-1206	**FYI on Questions for New Internet Users**
RFC-1207	**FYI on Questions for Experienced Internet Users**
RFC-1212	**Concise MIB Definitions**
RFC-1213	**Management Information Base for the Management of TCP/IP Based Internets - MIB-II**
RFC-1225	**Post Office Protocol Version 3 (POP3)**
RFC-1250	**IAB Official Protocols; August 1991**

There is no reference for this entry.

This reference is not included in the current version of the system. It may be present in future updates. If you require this reference, please contact InfoMagic.

[RFC-1261] Williamson, S.; Nobile, L. Transiton of NIC services. 1991 September; 3 p. (Format: TXT=4488 bytes)

[RFC-1260] Not yet issued.

[RFC-1259] Kapor, M. Building the open road: The NREN as test-bed for the national public network. 1991 September; 23 p. (Format: TXT=62944 bytes)

[RFC-1258] Kantor, B. BSD Rlogin. 1991 September; 5 p. (Format: TXT=10763 bytes)

[RFC-1257] Partridge, C. Isochronous applications do not require jitter-controlled networks. 1991 September; 5 p. (Format: TXT=11075 bytes)

[RFC-1256] Deering, S.E.,ed. ICMP router discovery messages. 1991 September; 19 p. (Format: TXT=44628 bytes)

[RFC-1255] North American Directory Forum. Naming scheme for c=US. 1991 September; 25 p. (Format: TXT=53783 bytes) (Obsoletes RFC 1218)

[RFC-1254] Mankin, A.; Ramakrishnan, K.K.,eds. Gateway congestion control survey. 1991 August; 25 p. (Format: TXT=69793 bytes)

[RFC-1253] Baker, F.; Coltun, R. OSPF version 2: Management Information Base. 1991 August; 42 p. (Format: TXT=77232 bytes) (Obsoletes RFC 1252)

[RFC-1252] Baker, F.; Coltun, R. OSPF version 2: Management Information Base. 1991 August; 42 p. (Format: TXT=77250 bytes) (Obsoletes RFC 1248; Obsoleted by RFC 1253)

[RFC-1251] Malkin, G.S. Who's who in the internet: Biographies of IAB, IESG and IRSG members. 1991 August; 26 p. (Format: TXT=72721 bytes) (Also FYI 9)

[RFC-1250] Postel, J.B.,ed. IAB official protocol standards. 1991 August; 28 p. (Format: TXT=65279 bytes) (Obsoletes RFC 1200)

[RFC-1249] Howes, T.; Smith, M.; Beecher, B. DIXIE protocol specification. 1991 August; 10 p. (Format: TXT=20693 bytes)

[RFC-1248] Baker, F.; Coltun, R. OSPF version 2: Management Information Base. 1991 July; 42 p. (Format: TXT=77126 bytes)

[RFC-1247] Moy, J. OSPF version 2. 1991 July; 189 p. (Format: PS=1063028, TXT=443917 bytes) (Obsoletes RFC 1131)

[RFC-1246] Moy, J.,ed. Experience with the OSPF protocol. 1991 July; 31 p. (Format: PS=146913, TXT=72180 bytes)

[RFC-1245] Moy, J.,ed. OSPF protocol analysis. 1991 July; 12 p. (Format: PS=64094 , TXT=27492 bytes)

[RFC-1244] Holbrook, J.P.; Reynolds, J.K.,eds. Site Security Handbook. 1991 July; 101 p. (Format: TXT=259129 bytes) (Also FYI 8)

[RFC-1243] Waldbusser, S.,ed. Appletalk Management Information Base. 1991 July; 29 p. (Format: TXT=61985 bytes)

[RFC-1242] Bradner, S.,ed. Benchmarking terminology for network interconnection devices. 1991 July; 12 p. (Format: TXT=22817 bytes)

[RFC-1241] Woodburn, R.A.; Mills, D.L. Scheme for an internet encapsulation protocol: Version 1. 1991 July; 17 p. (Format: TXT=42468, PS=128921 bytes)

[RFC-1240] Shue, C.; Haggerty, W.; Dobbins, K. OSI connectionless transport services on top of UDP: Version 1. 1991 June; 8 p. (Format: TXT=18140 bytes)

[RFC-1239] Reynolds, J.K. Reassignment of experimental MIBs to standard MIBs. 1991 June; 2 p. (Format: TXT=3656 bytes) (Updates RFC 1229, RFC 1230, RFC 1231, RFC 1232, RFC 1233)

[RFC-1238] Satz, G. CLNS MIB for use with Connectionless Network Protocol (ISO 8473) and End System to Intermediate System (ISO 9542). 1991 June; 32 p. (Format: TXT=65159 bytes) (Obsoletes RFC 1162)

[RFC-1237] Not yet issued.

[RFC-1236] Morales, L.F., Jr.; Hasse, P.R. IP to X.121 address mapping for DDN. 1991 June; 7 p. (Format: TXT=12626 bytes)

[RFC-1235] Ioannidis, J.; Maguire, G.Q., Jr. Coherent File Distribution Protocol. 1991 June; 12 p. (Format: TXT=29345 bytes)

[RFC-1234] Provan, D. Tunneling IPX traffic through IP networks. 1991 June; 6 p. (Format: TXT=12333 bytes)

[RFC-1233] Cox, T.A.; Tesink, K.,eds. Definitions of managed objects for the DS3 Interface type. 1991 May; 23 p. (Format: TXT=49559 bytes)

[RFC-1232] Baker, F.; Kolb, C.P.,eds. Definitions of managed objects for the DS1 Interface type. 1991 May; 28 p. (Format: TXT=60757 bytes)

[RFC-1231] McCloghrie, K.; Fox, R.; Decker, E. IEEE 802.5 Token Ring MIB. 1991 May; 23 p. (Format: TXT=53542 bytes)

[RFC-1230] McCloghrie, K.; Fox, R. IEEE 802.4 Token Bus MIB. 1991 May; 23 p. (Format: TXT=53100 bytes)

[RFC-1229] McCloghrie, K.,ed. Extensions to the generic-interface MIB. 1991 May; 16 p. (Format: TXT=36022 bytes)

[RFC-1228] Carpenter, G.; Wijnen, B. SNMP-DPI: Simple Network Management Protocol Distributed Program Interface. 1991 May; 50 p. (Format: TXT=96972 bytes)

[RFC-1227] Rose, M.T. SNMP MUX protocol and MIB. 1991 May; 13 p. (Format: TXT=25868 bytes)

[RFC-1226] Kantor, B. Internet protocol encapsulation of AX.25 frames. 1991 May; 2 p. (Format: TXT=2573 bytes)

[RFC-1225] Rose, M.T. Post Office Protocol: Version 3. 1991 May; 16 p. (Format: TXT=37340 bytes) (Obsoletes RFC 1081)

[RFC-1224] Steinberg, L. Techniques for managing asynchronously generated alerts. 1991 May; 22 p. (Format: TXT=54303 bytes)

[RFC-1223] Halpern, J.M. OSI CLNS and LLC1 protocols on Network Systems HYPERchannel. 1991 May; 12 p. (Format: TXT=29601 bytes)

[RFC-1222] Braun, H.W.; Rekhter, Y. Advancing the NSFNET routing architecture. 1991 May; 6 p. (Format: TXT=15067 bytes)

[RFC-1221] Edmond, W. Host Access Protocol (HAP) specification: Version 2. 1991 April; 68 p. (Format: TXT=156550 bytes) (Updates RFC 907)

[RFC-1220] Baker, F.,ed. Point-to-Point Protocol extensions for bridging. 1991 April; 18 p. (Format: TXT=38165 bytes)

[RFC-1219] Tsuchiya, P.F. On the assignment of subnet numbers. 1991 April; 13 p. (Format: TXT=30609 bytes)

[RFC-1218] North American Directory Forum. Naming scheme for c=US. 1991 April; 23 p. (Format: TXT=42698 bytes)

[RFC-1217] Cerf, V.G. Memo from the Consortium for Slow Commotion Research (CSCR). 1991 April 1; 5 p. (Format: TXT=11079 bytes)

[RFC-1216] Richard, P.; Kynikos, P. Gigabit network economics and paradigm shifts. 1991 April 1; 4 p. (Format: TXT=8130 bytes)

[RFC-1215] Rose, M.T.,ed. Convention for defining traps for use with the SNMP. 1991 March; 9 p. (Format: TXT=19336 bytes)

[RFC-1214] LaBarre, L.,ed. OSI internet management: Management Information Base. 1991 April; 83 p. (Format: TXT=172564 bytes)

[RFC-1213] McCloghrie, K.; Rose, M.T.,eds. Management Information Base for network management of TCP/IP-based internets:MIB-II. 1991 March; 70 p. (Format: TXT=146080 bytes) (Obsoletes RFC 1158)

[RFC-1212] Rose, M.T.; McCloghrie, K.,eds. Concise MIB definitions. 1991 March; 19 p. (Format: TXT=43579 bytes)

[RFC-1211] Westine, A.; Postel, J.B. Problems with the maintenance of large mailing lists. 1991 March; 54 p. (Format: TXT=96167 bytes)

[RFC-1210] Cerf, V.G.; Kirstein, P.T.; Randell, B.,eds. Network and infrastructure user requirements for transatlantic research collaboration: Brussels, July 16-18, and Washington July 24-25, 1990. 1991 March; 36 p. (Format: TXT=79048 bytes)

[RFC-1209] Piscitello, D.M.; Lawrence, J. Transmission of IP datagrams over the SMDS Service. 1991 March; 11 p. (Format: TXT=25280 bytes)

[RFC-1208] Jacobsen, O.J.; Lynch, D.C. Glossary of networking terms. 1991 March; 18 p. (Format: TXT=41156 bytes)

[RFC-1207] Malkin, G.; Marine, A.N.; Reynolds, J.K. FYI on Questions and Answers: Answers to commonly asked "experienced Internet user" questions. 1991 February; 15 p. (Format: TXT=33385 bytes) (Also FYI 7)

[RFC-1206] Malkin, G.S.; Marine, A.N. FYI on Questions and Answers: Answers to commonly asked "new Internet user" questions. 1991 February; 32 p. (Format: TXT=72479 bytes) (Also FYI 4) (Obsoletes RFC 1177)

[RFC-1205] Chmielewski, P. 5250 Telnet interface. 1991 February; 12 p. (Format: TXT=27179 bytes)

[RFC-1204] Yeh, S.; Lee, D. Message Posting Protocol (MPP). 1991 February; 6 p. (Format: TXT=11371 bytes)

[RFC-1203] Rice, J. Interactive Mail Access Protocol - version 3. 1991 February; 49 p. (Format: TXT=123325 bytes) (Obsoletes RFC 1064)

[RFC-1202] Rose, M.T. Directory Assistance service. 1991 February; 11 p. (Format: TXT=21645 bytes)

[RFC-1201] Provan, D. Transmitting IP traffic over ARCNET networks. 1991 February; 7 p. (Format: TXT=16959 bytes) (Obsoletes RFC 1051)

[RFC-1200] Defense Advanced Research Projects Agency, Internet Activities Board. IAB Official protocol standards. 1991 April; 31 p. (Format: TXT=62501 bytes) (Obsoletes RFC-1140)

[RFC-1198] Scheifler, R.W. FYI on the X window system. 1991 January; 3 p. (Format: TXT=3629 bytes) (Also FYI 6)

[RFC-1197] Sherman, M. Using ODA for translating multimedia information. 1990 December; 2 p. (Format: TXT=3620 bytes)

[RFC-1196] Zimmerman, D.P. Finger User Information Protocol. 1990 December; 12 p. (Format: TXT=24799 bytes) (Obsoletes RFC 1194)

[RFC-1195] Callon, R.W. Use of OSI IS-IS for routing in TCP/IP and dual environments. 1990 December; 65 p. (Format: PS=381799, TXT=192628 bytes)

[RFC-1194] Zimmerman, D.P. Finger User Information Protocol. 1990 November; 12 p. (Format: TXT=24626 bytes) (Obsoletes RFC 742; Obsoleted by RFC 1196)

[RFC-1193] Ferrari, D. Client requirements for real-time communication services. 1990 November; 24 p. (Format: TXT=61540 bytes)

[RFC-1192] Kahin, B.,ed. Commercialization of the Internet summary report. 1990 November; 13 p. (Format: TXT=35253 bytes)

[RFC-1191] Mogul, J.C.; Deering, S.E. Path MTU discovery. 1990 November; 19 p. (Format: TXT=47936 bytes) (Obsoletes RFC 1063)

[RFC-1190] Topolcic, C.,ed. Experimental Internet Stream Protocol, version 2 (ST-II). 1990 October; 148 p. (Format: TXT=386909 bytes) (Obsoletes IEN 119)

[RFC-1189] Warrier, U.S.; Besaw, L.; LaBarre, L.; Handspicker, B.D. Common Management Information Services and Protocols for the Internet (CMOT) and (CMIP). 1990 October; 15 p. (Format: TXT=32928 bytes) (Obsoletes RFC 1095)

[RFC-1188] Katz, D. Proposed standard for the transmission of IP datagrams over FDDI networks. 1990 October; 11 p. (Format: TXT=22424 bytes) (Obsoletes RFC 1103)

[RFC-1187] Rose, M.T.; McCloghrie, K.; Davin, J.R. Bulk table retrieval with the SNMP. 1990 October; 12 p. (Format: TXT=27220 bytes)

[RFC-1186] Rivest, R.L. MD4 message digest algorithm. 1990 October; 18 p. (Format: TXT=35391 bytes)

[RFC-1185] Jacobson, V.; Braden, R.T.; Zhang, L. TCP extension for high-speed paths. 1990 October; 21 p. (Format: TXT=49508 bytes)

[RFC-1184] Borman, D.A.,ed. Telnet Linemode option. 1990 October; 23 p. (Format: TXT=53085 bytes) (Obsoletes RFC 1116)

[RFC-1183] Everhart, C.F.; Mamakos, L.A.; Ullmann, R.; Mockapetris, P.V. New DNS RR definitions. 1990 October; 11 p. (Format: TXT=23788 bytes) (Updates RFC 1034, RFC 1035)

[RFC-1182] Not yet issued.

[RFC-1181] Blokzijl, R. RIPE terms of reference. September 1990; 2 p. (Format: TXT=2523 bytes)

[RFC-1180] Socolofsky, T.J.; Kale, C.J. TCP/IP tutorial. 1991 January; 28 p. (Format: TXT=65494 bytes)

[RFC-1179] Line printer daemon protocol. 1990 August; 14 p. (Format: TXT=24324 bytes)

[RFC-1178] Libes, D. Choosing a name for your computer. 1990 August; 8 p. (Format: TXT=18472 bytes) (Also FYI 5)

[RFC-1177] Malkin, G.S.; Marine, A.N.; Reynolds, J.K. FYI on Questions and Answers: Answers to commonly asked "new internet user" questions. 1990 August; 24 p. (Format: TXT=52852 bytes) (Also FYI 4)

[RFC-1176] Crispin, M.R. Interactive Mail Access Protocol - version 2. 1990 August; 30 p. (Format: TXT=67330 bytes) (Obsoletes RFC 1064)

[RFC-1175] Bowers, K.L.; LaQuey, T.L.; Reynolds, J.K.; Roubicek, K.; Stahl, M.K.; Yuan, A. FYI on where to start: A bibliography of internetworking information. 1990 August; 42 p. (Format: TXT=67330 bytes) (Also FYI 3)

[RFC-1174] Cerf, V.G. IAB recommended policy on distributing internet identifier assignment and IAB recommended policy change to internet "connected" status. 1990 August; 9 p. (Format: TXT=21321 bytes)

[RFC-1173] VanBokkelen, J. Responsibilities of host and network managers: A summary of the "oral tradition" of the Internet. 1990 August; 5 p. (Format: TXT=12527 bytes)

[RFC-1172] Perkins, D.; Hobby, R. Point-to-Point Protocol (PPP) initial configuration options. 1990 July; 38 p. (Format: TXT=76132 bytes)

[RFC-1171] Perkins, D. Point-to-Point Protocol for the transmission of multi-protocol datagrams over Point-to-Point links. 1990 July; 48 p. (Format: TXT=92321 bytes) (Obsoletes RFC 1134)

[RFC-1170] Fougner, R.B. Public key standards and licenses. 1991 January; 2 p. (Format: TXT=3144 bytes)

[RFC-1169] Cerf, V.G.; Mills, K.L. Explaining the role of GOSIP. 1990 August; 15 p. (Format: TXT=30255 bytes)

[RFC-1168] Westine, A.; DeSchon, A.L.; Postel, J.B.; Ward, C.E. Intermail and Commercial Mail Relay services. 1990 July; 23 p. (Format: PS=149816 bytes)

[RFC-1167] Cerf, V.G. Thoughts on the National Research and Education Network. 1990 July; 8 p. (Format: TXT=20682 bytes)

[RFC-1166] Kirkpatrick, S.; Stahl, M.K.; Recker, M. Internet numbers. 1990 July; 182 p. (Format: TXT=566778 bytes) (Obsoletes RFC 1117, RFC 1062, RFC 1020)

[RFC-1165] Crowcroft, J.; Onions, J.P. Network Time Protocol (NTP) over the OSI Remote Operations Service. 1990 June; 10 p. (Format: TXT=18277 bytes)

[RFC-1164] Honig, J.C.; Katz, D.; Mathis, M.; Rekhter, Y.; Yu, J.Y. Application of the Border Gateway Protocol in the Internet. 1990 June; 23 p. (Format: TXT=56278 bytes)

[RFC-1163] Lougheed, K.; Rekhter, Y. Border Gateway Protocol (BGP). 1990 June; 29 p. (Format: TXT=69404 bytes) (Obsoletes RFC 1105)

[RFC-1162] Satz, G. Connectionless Network Protocol (ISO 8473) and End System to Intermediate System (ISO 9542) Management Information Base. 1990 June; 70 p. (Format: TXT=109893 bytes)

[RFC-1161] Rose, M.T. SNMP over OSI. 1990 June; 8 p. (Format: TXT=16036 bytes)

[RFC-1160] Cerf, V. Internet Activities Board. 1990 May; 11 p. (Format: TXT=28182 bytes) (Obsoletes RFC 1120)

[RFC-1159] Nelson, R. Message Send Protocol. 1990 June; 2 p. (Format: TXT=3957 bytes)

[RFC-1158] Rose, M.T.,ed. Management Information Base for network management of TCP/IP-based internets: MIB-II. 1990 May; 133 p. (Format: TXT=212152 bytes)

[RFC-1157] Case, J.D.; Fedor, M.; Schoffstall, M.L.; Davin, C. Simple Network Management Protocol (SNMP). 1990 May; 36 p. (Format: TXT=74894 bytes) (Obsoletes RFC 1098)

[RFC-1156] McCloghrie, K.; Rose, M.T. Management Information Base for network management of TCP/IP-based internets. 1990 May; 91 p. (Format: TXT=138781 bytes) (Obsoletes RFC 1066)

[RFC-1155] Rose, M.T.; McCloghrie, K. Structure and identification of management information for TCP/IP-based internets. 1990 May; 22 p. (Format: TXT=40927 bytes) (Obsoletes RFC 1065)

[RFC-1154] Robinson, D.; Ullmann, R. Encoding header field for internet messages. 1990 April; 7 p. (Format: TXT=12214 bytes)

[RFC-1153] Wancho, F.J. Digest message format. 1990 April; 4 p. (Format: TXT=6632 bytes)

[RFC-1152] Partridge, C. Workshop report: Internet research steering group workshop on very-high-speed networks. 1990 April; 23 p. (Format: TXT=64003 bytes)

[RFC-1151] Partridge, C.; Hinden, R.M. Version 2 of the Reliable Data Protocol (RDP). 1990 April; 4 p. (Format: TXT=8293 bytes) (Updates RFC 908)

[RFC-1150] Malkin, G.S.; Reynolds, J.K. F.Y.I. on F.Y.I.: Introduction to the F.Y.I. notes. 1990 March; 4 p. (Format: TXT=7867 bytes) (Also FYI 1)

[RFC-1149] Waitzman, D. Standard for the transmission of IP datagrams on avian carriers. 1990 April 1; 2 p. (Format: TXT=3329 bytes)

[RFC-1148] Kille, S.E. Mapping between X.400(1988) / ISO 10021 and RFC 822. 1990 March; 94 p. (Format: TXT=194292 bytes) (Updates RFC 822, RFC 987, RFC 1026, RFC 1138)

[RFC-1147] Stine, R.H.,ed. FYI on a network management tool catalog: Tools for monitoring and debugging TCP/IP internets and interconnected devices. 1990 April; 126 p. (Format: TXT=336906, PS=555225 bytes) (Also FYI 2)

[RFC-1146] Zweig, J.; Partridge, C. TCP alternate checksum options. 1990 March; 5 p. (Format: TXT=10955 bytes) (Obsoletes RFC 1145)

[RFC-1145] Zweig, J.; Partridge, C. TCP alternate checksum options. 1990 February; 5 p. (Format: TXT=11052 bytes) (Obsoleted by RFC 1146)

[RFC-1144] Jacobson, V. Compressing TCP/IP headers for low-speed serial links. 1990 February; 43 p. (Format: TXT=120959, PS=534729 bytes)

[RFC-1143] Bernstein, D.J. Q method of implementing Telnet option negotiation. 1990 February; 10 p. (Format: TXT=23331 bytes)

[RFC-1142] Not yet issued.

[RFC-1141] Mallory, T.; Kullberg, A. Incremental updating of the Internet checksum. 1990 January; 2 p. (Format: TXT=3587 bytes) (Updates RFC 1071)

[RFC-1140] Defense Advanced Research Projects Agency, Internet Activities Board. IAB official protocol standards. 1990 May; 27 p. (Format: TXT=60501 bytes) (Obsoletes RFC 1130)

[RFC-1139] Hagens, R.A. Echo function for ISO 8473. 1990 January; 6 p. (Format: TXT=14229 bytes)

[RFC-1138] Kille, S.E. Mapping between X.400(1988) / ISO 10021 and RFC 822. 1989 December; 92 p. (Format: TXT=191029 bytes) (Updates RFC 822, RFC 987, RFC 1026; Updated by RFC 1148)

[RFC-1137] Kille, S.E. Mapping between full RFC 822 and RFC 822 with restricted encoding. 1989 December; 3 p. (Format: TXT=6436 bytes) (Updates RFC 976)

[RFC-1136] Hares, S.; Katz, D. Administrative Domains and Routing Domains: A model for routing in the Internet. 1989 December; 10 p. (Format: TXT=22158 bytes)

[RFC-1135] Reynolds, J.K. Helminthiasis of the Internet. 1989 December; 33 p. (Format: TXT=77033 bytes)

[RFC-1134] Perkins, D. Point-to-Point Protocol: A proposal for multi-protocol transmission of datagrams over Point-to-Point links. 1989 November; 38 p. (Format: TXT=87352 bytes) (Obsoleted by RFC 1171)

[RFC-1133] Yu, J.Y.; Braun, H.W. Routing between the NSFNET and the DDN. 1989 November; 10 p. (Format: TXT=23169 bytes)

[RFC-1132] McLaughlin, L.J. Standard for the transmission of 802.2 packets over IPX networks. 1989 November; 4 p. (Format: TXT=8128 bytes)

[RFC-1131] Moy, J. OSPF specification. 1989 October; 107 p. (Format: PS=857280 bytes)

[RFC-1130] Defense Advanced Research Projects Agency, Internet Activities Board. IAB official protocol standards. 1989 October; 17 p. (Format: TXT=33858 bytes) (Obsoletes RFC 1100; Obsoleted by RFC 1140)

[RFC-1129] Mills, D.L. Internet time synchronization: The Network Time Protocol. 1989 October; 29 p. (Format: PS=551697 bytes)

[RFC-1128] Mills, D.L. Measured performance of the Network Time Protocol in the Internet system. 1989 October; 20 p. (Format: PS=633742 bytes)

[RFC-1127] Braden, R.T. Perspective on the Host Requirements RFCs. 1989 October; 20 p. (Format: TXT=41267 bytes)

[RFC-1126] Little, M. Goals and functional requirements for inter-autonomous system routing. 1989 October; 25 p. (Format: TXT=62725 bytes)

[RFC-1125] Estrin, D. Policy requirements for inter Administrative Domain routing. 1989 November; 18 p. (Format: TXT=55248, PS=282123 bytes)

[RFC-1124] Leiner, B.M. Policy issues in interconnecting networks. 1989 September; 54 p. (Format: PS=315692 bytes)

[RFC-1123] Braden, R.T.,ed. Requirements for Internet hosts - application and support. 1989 October; 98 p. (Format: TXT=245503 bytes)

[RFC-1122] Braden, R.T.,ed. Requirements for Internet hosts - communication layers. 1989 October; 116 p. (Format: TXT=295992 bytes)

[RFC-1121] Postel, J.B.; Kleinrock, L.; Cerf, V.G.; Boehm, B. Act one - the poems. 1989 September; 6 p. (Format: TXT=10644 bytes)

[RFC-1120] Cerf, V. Internet Activities Board. 1989 September; 11 p. (Format: TXT=26123 bytes) (Obsoleted by RFC 1160)

[RFC-1119] Mills, D.L. Network Time Protocol (version 2) specification and implementation. 1989 September; 64 p. (Format: PS=535202 bytes) (Obsoletes RFC 1059, RFC 958)

[RFC-1118] Krol, E. Hitchhikers guide to the Internet. 1989 September; 24 p. (Format: TXT=62757 bytes)

[RFC-1117] Romano, S.; Stahl, M.K.; Recker, M. Internet numbers. 1989 August; 109 p. (Format: TXT=324666 bytes) (Obsoletes RFC 1062, RFC 1020, RFC 997; Obsoleted by RFC 1166)

[RFC-1116] Borman, D.A.,ed. Telnet Linemode option. 1989 August; 21 p. (Format: TXT=47473 bytes) (Obsoleted by RFC 1184)

[RFC-1115] Linn, J. Privacy enhancement for Internet electronic mail: Part III - algorithms, modes, and identifiers [Draft]. 1989 August; 8 p. (Format: TXT=18226 bytes)

[RFC-1114] Kent, S.T.; Linn, J. Privacy enhancement for Internet electronic mail: Part II - certificate-based key management [Draft]. 1989 August; 25 p. (Format: TXT=69661 bytes)

[RFC-1113] Linn, J. Privacy enhancement for Internet electronic mail: Part I - message encipherment and authentication procedures [Draft]. 1989 August; 34 p. (Format: TXT=89293 bytes) (Obsoletes RFC 989, RFC 1040)

[RFC-1112] Deering, S.E. Host extensions for IP multicasting. 1989 August; 17 p. (Format: TXT=39904 bytes) (Obsoletes RFC 988, RFC 1054)

[RFC-1111] Postel, J.B. Request for comments on Request for Comments: Instructions to RFC authors. 1989 August; 6 p. (Format: TXT=11793 bytes) (Obsoletes RFC 825)

[RFC-1110] McKenzie, A.M. Problem with the TCP big window option. 1989 August; 3 p. (Format: TXT=5778 bytes)

[RFC-1109] Cerf, V.G. Report of the second Ad Hoc Network Management Review Group. 1989 August; 8 p. (Format: TXT=20642 bytes)

[RFC-1108] Not yet issued.

[RFC-1107] Sollins, K.R. Plan for Internet directory services. 1989 July; 19 p. (Format: TXT=51773 bytes)

[RFC-1106] Fox, R. TCP big window and NAK options. 1989 June; 13 p. (Format: TXT=37105 bytes)

[RFC-1105] Lougheed, K.; Rekhter, Y. Border Gateway Protocol (BGP). 1989 June; 17 p. (Format: TXT=37644 bytes) (Obsoleted by RFC 1163)

[RFC-1104] Braun, H.W. Models of policy based routing. 1989 June; 10 p. (Format: TXT=25468 bytes)

[RFC-1103] Katz, D. Proposed standard for the transmission of IP datagrams over FDDI Networks. 1989 June; 9 p. (Format: TXT=19439 bytes) (Obsoleted by RFC 1188)

[RFC-1102] Clark, D.D. Policy routing in Internet protocols. 1989 May; 22 p. (Format: TXT=59664 bytes)

[RFC-1101] Mockapetris, P.V. DNS encoding of network names and other types. 1989 April; 14 p. (Format: TXT=28677 bytes) (Updates RFC 1034, RFC 1035)

[RFC-1100] Defense Advanced Research Projects Agency, Internet Activities Board. IAB official protocol standards. 1989 April; 14 p. (Format: TXT=30101 bytes) (Obsoletes RFC 1083; Obsoleted by RFC 1130)

[RFC-1099] Not yet issued.

[RFC-1098] Case, J.D.; Fedor, M.; Schoffstall, M.L.; Davin, C. Simple Network Management Protocol (SNMP). 1989 April; 34 p. (Format: TXT=71563 bytes) (Obsoletes RFC 1067; Obsoleted by RFC 1157)

[RFC-1097] Miller, B. Telnet subliminal-message option. 1989 April 1; 3 p. (Format: TXT=5490 bytes)

[RFC-1096] Marcy, G.A. Telnet X display location option. 1989 March; 3 p. (Format: TXT=4634 bytes)

[RFC-1095] Warrier, U.S.; Besaw, L. Common Management Information Services and Protocol over TCP/IP (CMOT). 1989 April; 67 p. (Format: TXT=157506 bytes) (Obsoleted by RFC 1189)

[RFC-1094] Sun Microsystems, Inc. NFS: Network File System Protocol specification. 1989 March; 27 p. (Format: TXT=51454 bytes)

[RFC-1093] Braun, H.W. NSFNET routing architecture. 1989 February; 9 p. (Format: TXT=20629 bytes)

[RFC-1092] Rekhter, J. EGP and policy based routing in the new NSFNET backbone. 1989 February; 5 p. (Format: TXT=11865 bytes)

[RFC-1091] VanBokkelen, J. Telnet terminal-type option. 1989 February; 7 p. (Format: TXT=13439 bytes) (Obsoletes RFC 930)

[RFC-1090] Ullmann, R. SMTP on X.25. 1989 February; 4 p. (Format: TXT=6141 bytes)

[RFC-1089] Schoffstall, M.L.; Davin, C.; Fedor, M.; Case, J.D. SNMP over Ethernet. 1989 February; 3 p. (Format: TXT=4458 bytes)

[RFC-1088] McLaughlin, L.J. Standard for the transmission of IP datagrams over NetBIOS networks. 1989 February; 3 p. (Format: TXT=5749 bytes)

[RFC-1087] Defense Advanced Research Projects Agency, Internet Activities Board. Ethics and the Internet. 1989 January; 2 p. (Format: TXT=4582 bytes)

[RFC-1086] Onions, J.P.; Rose, M.T. ISO-TP0 bridge between TCP and X.25. 1988 December; 9 p. (Format: TXT=19934 bytes)

[RFC-1085] Rose, M.T. ISO presentation services on top of TCP/IP based internets. 1988 December; 32 p. (Format: TXT=64643 bytes)

[RFC-1084] Reynolds, J.K. BOOTP vendor information extensions. 1988 December; 8 p. (Format: TXT=16327 bytes) (Obsoletes RFC 1048)

[RFC-1083] Defense Advanced Research Projects Agency, Internet Activities Board. IAB official protocol standards. 1988 December; 12 p. (Format: TXT=27128 bytes) (Obsoleted by RFC 1100)

[RFC-1082] Rose, M.T. Post Office Protocol - version 3: Extended service offerings. 1988 November; 11 p. (Format: TXT=25423 bytes)

[RFC-1081] Rose, M.T. Post Office Protocol - version 3. 1988 November; 16 p. (Format: TXT=37009 bytes)

[RFC-1080] Hedrick, C.L. Telnet remote flow control option. 1988 November; 4 p. (Format: TXT=6688 bytes)

[RFC-1079] Hedrick, C.L. Telnet terminal speed option. 1988 December; 3 p. (Format: TXT=4942 bytes)

[RFC-1078] Lottor, M. TCP port service Multiplexer (TCPMUX). 1988 November; 2 p. (Format: TXT=3248 bytes)

[RFC-1077] Leiner, B.M.,ed. Critical issues in high bandwidth networking. 1988 November; 46 p. (Format: TXT=116464 bytes)

[RFC-1076] Trewitt, G.; Partridge, C. HEMS monitoring and control language. 1988 November; 42 p. (Format: TXT=98774 bytes) (Obsoletes RFC 1023)

[RFC-1075] Waitzman, D.; Partridge, C.; Deering, S.E. Distance Vector Multicast Routing Protocol. 1988 November; 24 p. (Format: TXT=54731 bytes)

[RFC-1074] Rekhter, J. NSFNET backbone SPF based Interior Gateway Protocol. 1988 October; 5 p. (Format: TXT=10872 bytes)

[RFC-1073] Waitzman, D. Telnet window size option. 1988 October; 4 p. (Format: TXT=7639 bytes)

[RFC-1072] Jacobson, V.; Braden, R.T. TCP extensions for long-delay paths. 1988 October; 16 p. (Format: TXT=36000 bytes)

[RFC-1071] Braden, R.T.; Borman, D.A.; Partridge, C. Computing the Internet checksum. 1988 September; 24 p. (Format: TXT=54941 bytes) (Updated by RFC 1141)

[RFC-1070] Hagens, R.A.; Hall, N.E.; Rose, M.T. Use of the Internet as a subnetwork for experimentation with the OSI network layer. 1989 February; 17 p. (Format: TXT=37354 bytes)

[RFC-1069] Callon, R.W.; Braun, H.W. Guidelines for the use of Internet-IP addresses in the ISO Connectionless-Mode Network Protocol. 1989 February; 10 p. (Format: TXT=24268 bytes) (Obsoletes RFC 986)

[RFC-1068] DeSchon, A.L.; Braden, R.T. Background File Transfer Program (BFTP). 1988 August; 27 p. (Format: TXT=51004 bytes)

[RFC-1067] Case, J.D.; Fedor, M.; Schoffstall, M.L.; Davin, J. Simple Network Management Protocol. 1988 August; 33 p. (Format: TXT=69592 bytes) (Obsoleted by RFC 1098)

[RFC-1066] McCloghrie, K.; Rose, M.T. Management Information Base for network management of TCP/IP-based internets. 1988 August; 90 p. (Format: TXT=135177 bytes) (Obsoleted by RFC 1156)

[RFC-1065] McCloghrie, K.; Rose, M.T. Structure and identification of management information for TCP/IP-based internets. 1988 August; 21 p. (Format: TXT=38858 bytes) (Obsoleted by RFC 1155)

[RFC-1064] Crispin, M. Interactive Mail Access Protocol: Version 2. 1988 July; 26 p. (Format: TXT=57813 bytes) (Obsoleted by RFC 1176)

[RFC-1063] Mogul, J.C.; Kent, C.A.; Partridge, C.; McCloghrie, K. IP MTU discovery options. 1988 July; 11 p. (Format: TXT=27121 bytes) (Obsoleted by RFC 1191)

[RFC-1062] Romano, S.; Stahl, M.K.; Recker, M. Internet numbers. 1988 August; 65 p. (Format: TXT=198729 bytes) (Obsoletes RFC 1020; Obsoleted by RFC 1117)

[RFC-1061] Not yet issued.

[RFC-1060] Reynolds, J.K.; Postel, J.B. Assigned numbers. 1990 March; 86 p. (Format: TXT=177923 bytes) (Obsoletes RFC 1010)

[RFC-1059] Mills, D.L. Network Time Protocol (version 1) specification and implementation. 1988 July; 58 p. (Format: TXT=140890 bytes) (Obsoleted by RFC 1119)

[RFC-1058] Hedrick, C.L. Routing Information Protocol. 1988 June; 33 p. (Format: TXT=93285 bytes)

[RFC-1057] Sun Microsystems, Inc. RPC: Remote Procedure Call Protocol specification version 2. 1988 June; 25 p. (Format: TXT=52462 bytes) (Obsoletes RFC 1050)

[RFC-1056] Lambert, M.L. PCMAIL: A distributed mail system for personal computers. 1988 June; 38 p. (Format: TXT=85368 bytes) (Obsoletes RFC 993)

[RFC-1055] Romkey, J.L. Nonstandard for transmission of IP datagrams over serial lines: SLIP. 1988 June; 6 p. (Format: TXT=12911 bytes)

[RFC-1054] Deering, S.E. Host extensions for IP multicasting. 1988 May; 19 p. (Format: TXT=45465 bytes) (Obsoletes RFC 988; Obsoleted by RFC 1112)

[RFC-1053] Levy, S.; Jacobson, T. Telnet X.3 PAD option. 1988 April; 21 p. (Format: TXT=48952 bytes)

[RFC-1052] Cerf, V.G. IAB recommendations for the development of Internet network management standards. 1988 April; 14 p. (Format: TXT=30569 bytes)

[RFC-1051] Prindeville, P.A. Standard for the transmission of IP datagrams and ARP packets over ARCNET networks. 1988 March; 4 p. (Format: TXT=7779 bytes)

[RFC-1050] Sun Microsystems, Inc. RPC: Remote Procedure Call Protocol specification. 1988 April; 24 p. (Format: TXT=51540 bytes) (Obsoleted by RFC 1057)

[RFC-1049] Sirbu, M.A. Content-type header field for Internet messages. 1988 March; 8 p. (Format: TXT=18923 bytes)

[RFC-1048] Prindeville, P.A. BOOTP vendor information extensions. 1988 February; 7 p. (Format: TXT=15423 bytes) (Obsoleted by RFC 1084)

[RFC-1047] Partridge, C. Duplicate messages and SMTP. 1988 February; 3 p. (Format: TXT=5888 bytes)

[RFC-1046] Prue, W.; Postel, J.B. Queuing algorithm to provide type-of-service for IP links. 1988 February; 11 p. (Format: TXT=30106 bytes)

[RFC-1045] Cheriton, D.R. VMTP: Versatile Message Transaction Protocol: Protocol specification. 1988 February; 123 p. (Format: TXT=272058 bytes)

[RFC-1044] Hardwick, K.; Lekashman, J. Internet Protocol on Network System's HYPERchannel: Protocol specification. 1988 February; 43 p. (Format: TXT=103241 bytes)

[RFC-1043] Yasuda, A.; Thompson, T. Telnet Data Entry Terminal option: DODIIS implementation. 1988 February; 26 p. (Format: TXT=59478 bytes) (Updates RFC 732)

[RFC-1042] Postel, J.B.; Reynolds, J.K. Standard for the transmission of IP datagrams over IEEE 802 networks. 1988 February; 15 p. (Format: TXT=35201 bytes) (Obsoletes RFC 948)

[RFC-1041] Rekhter, Y. Telnet 3270 regime option. 1988 January; 6 p. (Format: TXT=11608 bytes)

[RFC-1040] Linn, J. Privacy enhancement for Internet electronic mail: Part I: Message encipherment and authentication procedures. 1988 January; 29 p. (Format: TXT=76276 bytes) (Obsoletes RFC 989; Obsoleted by RFC 1113)

[RFC-1039] Latham, D. DoD statement on Open Systems Interconnection protocols. 1988 January; 3 p. (Format: TXT=6194 bytes) (Obsoletes RFC 945)

[RFC-1038] St. Johns, M. Draft revised IP security option. 1988 January; 7 p. (Format: TXT=15879 bytes)

[RFC-1037] Greenberg, B.; Keene, S. NFILE - a file access protocol. 1987 December; 86 p. (Format: TXT=197312 bytes)

[RFC-1036] Horton, M.R.; Adams, R. Standard for interchange of USENET messages. 1987 December; 19 p. (Format: TXT=46891 bytes) (Obsoletes RFC 850)

[RFC-1035] Mockapetris, P.V. Domain names - implementation and specification. 1987 November; 55 p. (Format: TXT=125626 bytes) (Obsoletes RFC 973, RFC 882, RFC 883; Updated by RFC 1101, RFC 1183)

[RFC-1034] Mockapetris, P.V. Domain names - concepts and facilities. 1987 November; 55 p. (Format: TXT=129180 bytes) (Obsoletes RFC 973, RFC 882, RFC 883; Updated by RFC 1101, RFC 1183)

[RFC-1033] Lottor, M. Domain administrators operations guide. 1987 November; 22 p. (Format: TXT=37263 bytes)

[RFC-1032] Stahl, M.K. Domain administrators guide. 1987 November; 14 p. (Format: TXT=29454 bytes)

[RFC-1031] Lazear, W.D. MILNET name domain transition. 1987 November; 10 p. (Format: TXT=20137 bytes)

[RFC-1030] Lambert, M.L. On testing the NETBLT Protocol over divers networks. 1987 November; 16 p. (Format: TXT=40964 bytes)

[RFC-1029] Parr, G. More fault tolerant approach to address resolution for a Multi-LAN system of Ethernets. 1988 May; 17 p. (Format: TXT=44019 bytes)

[RFC-1028] Davin, J.; Case, J.D.; Fedor, M.; Schoffstall, M.L. Simple Gateway Monitoring Protocol. 1987 November; 38 p. (Format: TXT=82440 bytes)

[RFC-1027] Carl-Mitchell, S.; Quarterman, J.S. Using ARP to implement transparent subnet gateways. 1987 October; 8 p. (Format: TXT=21297 bytes)

[RFC-1026] Kille, S.E. Addendum to RFC 987: (Mapping between X.400 and RFC-822). 1987 September; 4 p. (Format: TXT=7117 bytes) (Updates RFC 987; Updated by RFC 1138, RFC 1148)

[RFC-1025] Postel, J.B. TCP and IP bake off. 1987 September; 6 p. (Format: TXT=11648 bytes)

[RFC-1024] Partridge, C.; Trewitt, G. HEMS variable definitions. 1987 October; 74 p. (Format: TXT=126536 bytes)

[RFC-1023] Trewitt, G.; Partridge, C. HEMS monitoring and control language. 1987 October; 17 p. (Format: TXT=40992 bytes) (Obsoleted by RFC 1076)

[RFC-1022] Partridge, C.; Trewitt, G. High-level Entity Management Protocol (HEMP). 1987 October; 12 p. (Format: TXT=25348 bytes)

[RFC-1021] Partridge, C.; Trewitt, G. High-level Entity Management System (HEMS). 1987 October; 5 p. (Format: TXT=12993 bytes)

[RFC-1020] Romano, S.; Stahl, M.K. Internet numbers. 1987 November; 51 p. (Format: TXT=146864 bytes) (Obsoletes RFC 997; Obsoleted by RFC 1062, RFC 1117)

[RFC-1019] Arnon, D. Report of the Workshop on Environments for Computational Mathematics. 1987 September; 8 p. (Format: TXT=21151 bytes)

[RFC-1018] McKenzie, A.M. Some comments on SQuID. 1987 August; 3 p. (Format: TXT=7931 bytes)

[RFC-1017] Leiner, B.M. Network requirements for scientific research: Internet task force on scientific computing. 1987 August; 19 p. (Format: TXT=49512 bytes)

[RFC-1016] Prue, W.; Postel, J.B. Something a host could do with source quench: The Source Quench Introduced Delay (SQuID). 1987 July; 18 p. (Format: TXT=47922 bytes)

[RFC-1015] Leiner, B.M. Implementation plan for interagency research Internet. 1987 July; 24 p. (Format: TXT=63159 bytes)

[RFC-1014] Sun Microsystems, Inc. XDR: External Data Representation standard. 1987 June; 20 p. (Format: TXT=39316 bytes)

[RFC-1013] Scheifler, R.W. X Window System Protocol, version 11: Alpha update April 1987. 1987 June; 101 p. (Format: TXT=244905 bytes)

[RFC-1012] Reynolds, J.K.; Postel, J.B. Bibliography of Request For Comments 1 through 999. 1987 June; 64 p. (Format: TXT=129194 bytes)

[RFC-1011] Reynolds, J.K.; Postel, J.B. Official Internet protocols. 1987 May; 52 p. (Format: TXT=74593 bytes) (Obsoletes RFC 991)

[RFC-1010] Reynolds, J.K.; Postel, J.B. Assigned numbers. 1987 May; 44 p. (Format: TXT=78179 bytes) (Obsoletes RFC 990; Obsoleted by RFC 1060)

[RFC-1009] Braden, R.T.; Postel, J.B. Requirements for Internet gateways. 1987 June; 55 p. (Format: TXT=128173 bytes) (Obsoletes RFC 985)

[RFC-1008] McCoy, W. Implementation guide for the ISO Transport Protocol. 1987 June; 73 p. (Format: TXT=204664 bytes)

[RFC-1007] McCoy, W. Military supplement to the ISO Transport Protocol. 1987 June; 23 p. (Format: TXT=51280 bytes)

[RFC-1006] Rose, M.T.; Cass, D.E. ISO transport services on top of the TCP: Version: 3. 1987 May; 17 p. (Format: TXT=31935 bytes) (Obsoletes RFC 983)

[RFC-1005] Khanna, A.; Malis, A.G. ARPANET AHIP-E Host Access Protocol (enhanced AHIP). 1987 May; 31 p. (Format: TXT=69957 bytes)

[RFC-1004] Mills, D.L. Distributed-protocol authentication scheme. 1987 April; 8 p. (Format: TXT=21402 bytes)

[RFC-1003] Katz, A.R. Issues in defining an equations representation standard. 1987 March; 7 p. (Format: TXT=19816 bytes)

[RFC-1002] Defense Advanced Research Projects Agency, Internet Activities Board, End-to-End Services Task Force, NetBIOS Working Group. Protocol standard for a NetBIOS service on a TCP/UDP transport: Detailed specifications. 1987 March; 85 p. (Format: TXT=170262 bytes)

[RFC-1001] Defense Advanced Research Projects Agency, Internet Activities Board, End-to-End Services Task Force, NetBIOS Working Group. Protocol standard for a NetBIOS service on a TCP/UDP transport: Concepts and methods. 1987 March; 68 p. (Format: TXT=158437 bytes)

[RFC-1000] Reynolds, J.K.; Postel, J.B. Request For Comments reference guide. 1987 August; 149 p. (Format: TXT=323960 bytes) (Obsoletes RFC 999)

[RFC-999] Westine, A.; Postel, J.B. Requests For Comments summary notes: 900-999. 1987 April; 22 p. (Format: TXT=62877 bytes) (Obsoleted by RFC 1000)

[RFC-998] Clark, D.D.; Lambert, M.L.; Zhang, L. NETBLT: A bulk data transfer protocol. 1987 March; 21 p. (Format: TXT=57147 bytes) (Obsoletes RFC 969)

[RFC-997] Reynolds, J.K.; Postel, J.B. Internet numbers. 1987 March; 42 p. (Format: TXT=123919 bytes) (Obsoleted by RFC 1020, RFC 1117; Updates RFC 990)

[RFC-996] Mills, D.L. Statistics server. 1987 February; 3 p. (Format: TXT=6127 bytes)

[RFC-995] International Organization for Standardization. End System to Intermediate System Routing Exchange Protocol for use in conjunction with ISO 8473. 1986 April; 41 p. (Format: TXT=94069 bytes)

[RFC-994] International Organization for Standardization. Final text of DIS 8473, Protocol for Providing the Connectionless-mode Network Service. 1986 March; 52 p. (Format: TXT=129006 bytes) (Obsoletes RFC 926)

[RFC-993] Clark, D.D.; Lambert, M.L. PCMAIL: A distributed mail system for personal computers. 1986 December; 28 p. (Format: TXT=71725 bytes) (Obsoletes RFC 984; Obsoleted by RFC 1056)

[RFC-992] Birman, K.P.; Joseph, T.A. On communication support for fault tolerant process groups. 1986 November; 18 p. (Format: TXT=52313 bytes)

[RFC-991] Reynolds, J.K.; Postel, J.B. Official ARPA-Internet protocols. 1986 November; 46 p. (Format: TXT=65205 bytes) (Obsoletes RFC 961; Obsoleted by RFC 1011)

[RFC-990] Reynolds, J.K.; Postel, J.B. Assigned numbers. 1986 November; 75 p. (Format: TXT=174784 bytes) (Obsoletes RFC 960; Obsoleted by RFC 1010; Updated by RFC 997)

[RFC-989] Linn, J. Privacy enhancement for Internet electronic mail: Part I: Message encipherment and authentication procedures. 1987 February; 23 p. (Format: TXT=63934 bytes) (Obsoleted by RFC 1040, RFC 1113)

[RFC-988] Deering, S.E. Host extensions for IP multicasting. 1986 July; 20 p. (Format: TXT=45220 bytes) (Obsoletes RFC 966; Obsoleted by RFC 1054, RFC 1112)

[RFC-987] Kille, S.E. Mapping between X.400 and RFC 822. 1986 June; 69 p. (Format: TXT=127540 bytes) (Updated by RFC 1026, RFC 1138, RFC 1148)

[RFC-986] Callon, R.W.; Braun, H.W. Guidelines for the use of Internet-IP addresses in the ISO Connectionless-Mode Network Protocol [Working draft]. 1986 June; 7 p. (Format: TXT=13950 bytes) (Obsoleted by RFC 1069)

[RFC-985] National Science Foundation, Network Technical Advisory Group. Requirements for Internet gateways - draft. 1986 May; 23 p. (Format: TXT=59221 bytes) (Obsoleted by RFC 1009)

[RFC-984] Clark, D.D.; Lambert, M.L. PCMAIL: A distributed mail system for personal computers. 1986 May; 31 p. (Format: TXT=69333 bytes) (Obsoleted by RFC 993)

[RFC-983] Cass, D.E.; Rose, M.T. ISO transport arrives on top of the TCP. 1986 April; 27 p. (Format: TXT=59819 bytes) (Obsoleted by RFC 1006)

[RFC-982] Braun, H.W. Guidelines for the specification of the structure of the Domain Specific Part (DSP) of the ISO standard NSAP address. 1986 April; 11 p. (Format: TXT=22595 bytes)

[RFC-981] Mills, D.L. Experimental multiple-path routing algorithm. 1986 March; 22 p. (Format: TXT=59069 bytes)

[RFC-980] Jacobsen, O.J.; Postel, J.B. Protocol document order information. 1986 March; 12 p. (Format: TXT=24416 bytes)

[RFC-979] Malis, A.G. PSN End-to-End functional specification. 1986 March; 15 p. (Format: TXT=39472 bytes)

[RFC-978] Reynolds, J.K.; Gillman, R.; Brackenridge, W.A.; Witkowski, A.; Postel, J.B. Voice File Interchange Protocol (VFIP). 1986 February; 5 p. (Format: TXT=9223 bytes)

[RFC-977] Kantor, B.; Lapsley, P. Network News Transfer Protocol. 1986 February; 27 p. (Format: TXT=55062 bytes)

[RFC-976] Horton, M.R. UUCP mail interchange format standard. 1986 February; 12 p. (Format: TXT=26814 bytes)

[RFC-975] Mills, D.L. Autonomous confederations. 1986 February; 10 p. (Format: TXT=28010 bytes)

[RFC-974] Partridge, C. Mail routing and the domain system. 1986 January; 7 p. (Format: TXT=18581 bytes)

[RFC-973] Mockapetris, P.V. Domain system changes and observations. 1986 January; 10 p. (Format: TXT=22364 bytes) (Obsoleted by RFC 1034, RFC 1035; Updates RFC 882, RFC 883)

[RFC-972] Wancho, F. Password Generator Protocol. 1986 January; 2 p. (Format: TXT=3890 bytes)

[RFC-971] DeSchon, A.L. Survey of data representation standards. 1986 January; 9 p. (Format: TXT=22883 bytes)

[RFC-970] Nagle, J. On packet switches with infinite storage. 1985 December; 9 p. (Format: TXT=24970 bytes)

[RFC-969] Clark, D.D.; Lambert, M.L.; Zhang, L. NETBLT: A bulk data transfer protocol. 1985 December; 15 p. (Format: TXT=40894 bytes) (Obsoleted by RFC 998)

[RFC-968] Cerf, V.G. Twas the night before start-up. 1985 December; 2 p. (Format: TXT=2573 bytes)

[RFC-967] Padlipsky, M.A. All victims together. 1985 December; 2 p. (Format: TXT=4820 bytes)

[RFC-966] Deering, S.E.; Cheriton, D.R. Host groups: A multicast extension to the Internet Protocol. 1985 December; 27 p. (Format: TXT=61006 bytes) (Obsoleted by RFC 988)

[RFC-965] Aguilar, L. Format for a graphical communication protocol. 1985 December; 51 p. (Format: TXT=108361 bytes)

[RFC-964] Sidhu, D.P. Some problems with the specification of the Military Standard Transmission Control Protocol. 1985 November; 10 p. (Format: TXT=21542 bytes)

[RFC-963] Sidhu, D.P. Some problems with the specification of the Military Standard Internet Protocol. 1985 November; 19 p. (Format: TXT=45102 bytes)

[RFC-962] Padlipsky, M.A. TCP-4 prime. 1985 November; 2 p. (Format: TXT=2885 bytes)

[RFC-961] Reynolds, J.K.; Postel, J.B. Official ARPA-Internet protocols. 1985 December; 38 p. (Format: TXT=54874 bytes) (Obsoletes RFC 944; Obsoleted by RFC 991)

[RFC-960] Reynolds, J.K.; Postel, J.B. Assigned numbers. 1985 December; 60 p. (Format: TXT=129292 bytes) (Obsoletes RFC 943; Obsoleted by RFC 990)

[RFC-959] Postel, J.B.; Reynolds, J.K. File Transfer Protocol. 1985 October; 69 p. (Format: TXT=151249 bytes) (Obsoletes RFC 765 [IEN 149])

[RFC-958] Mills, D.L. Network Time Protocol (NTP). 1985 September; 14 p. (Format: TXT=31520 bytes) (Obsoleted by RFC 1119)

[RFC-957] Mills, D.L. Experiments in network clock synchronization. 1985 September; 27 p. (Format: TXT=70490 bytes)

[RFC-956] Mills, D.L. Algorithms for synchronizing network clocks. 1985 September; 26 p. (Format: TXT=68868 bytes)

[RFC-955] Braden, R.T. Towards a transport service for transaction processing applications. 1985 September; 10 p. (Format: TXT=23066 bytes)

[RFC-954] Harrenstien, K.; Stahl, M.K.; Feinler, E.J. NICNAME/WHOIS. 1985 October; 4 p. (Format: TXT=7623 bytes) (Obsoletes RFC 812)

[RFC-953] Harrenstien, K.; Stahl, M.K.; Feinler, E.J. Hostname Server. 1985 October; 5 p. (Format: TXT=8588 bytes) (Obsoletes RFC 811)

[RFC-952] Harrenstien, K.; Stahl, M.K.; Feinler, E.J. DoD Internet host table specification. 1985 October; 6 p. (Format: TXT=12728 bytes) (Obsoletes RFC 810)

[RFC-951] Croft, W.J.; Gilmore, J. Bootstrap Protocol. 1985 September; 12 p. (Format: TXT=29038 bytes)

[RFC-950] Mogul, J.C.; Postel, J.B. Internet standard subnetting procedure. 1985 August; 18 p. (Format: TXT=39010 bytes) (Updates RFC 792)

[RFC-949] Padlipsky, M.A. FTP unique-named store command. 1985 July; 2 p. (Format: TXT=4130 bytes)

[RFC-948] Winston, I. Two methods for the transmission of IP datagrams over IEEE 802.3 networks. 1985 June; 5 p. (Format: TXT=11843 bytes) (Obsoleted by RFC 1042)

[RFC-947] Lebowitz, K.; Mankins, D. Multi-network broadcasting within the Internet. 1985 June; 5 p. (Format: TXT=12854 bytes)

[RFC-946] Nedved, R. Telnet terminal location number option. 1985 May; 4 p. (Format: TXT=6513 bytes)

[RFC-945] Postel, J.B. DoD statement on the NRC report. 1985 May; 2 p. (Format: TXT=5131 bytes) (Obsoleted by RFC 1039)

[RFC-944] Reynolds, J.K.; Postel, J.B. Official ARPA-Internet protocols. 1985 April; 40 p. (Format: TXT=63693 bytes) (Obsoletes RFC 924; Obsoleted by RFC 961)

[RFC-943] Reynolds, J.K.; Postel, J.B. Assigned numbers. 1985 April; 50 p. (Format: TXT=108133 bytes) (Obsoletes RFC 923; Obsoleted by RFC 960)

[RFC-942] National Research Council. Transport protocols for Department of Defense data networks. 1985 February; 68 p. (Format: TXT=222477 bytes)

[RFC-941] International Organization for Standardization. Addendum to the network service definition covering network layer addressing. 1985 April; 34 p. (Format: TXT=70706 bytes)

[RFC-940] Gateway Algorithms and Data Structures Task Force. Toward an Internet standard scheme for subnetting. 1985 April; 3 p. (Format: TXT=7061 bytes)

[RFC-939] National Research Council. Executive summary of the NRC report on transport protocols for Department of Defense data networks. 1985 February; 20 p. (Format: TXT=43485 bytes)

[RFC-938] Miller, T. Internet Reliable Transaction Protocol functional and interface specification. 1985 February; 16 p. (Format: TXT=40561 bytes)

[RFC-937] Butler, M.; Postel, J.B.; Chase, D.; Goldberger, J.; Reynolds, J.K. Post Office Protocol - version 2. 1985 February; 24 p. (Format: TXT=43762 bytes) (Obsoletes RFC 918)

[RFC-936] Karels, M.J. Another Internet subnet addressing scheme. 1985 February; 4 p. (Format: TXT=10407 bytes)

[RFC-935] Robinson, J.G. Reliable link layer protocols. 1985 January; 13 p. (Format: TXT=32335 bytes)

[RFC-934] Rose, M.T.; Stefferud, E.A. Proposed standard for message encapsulation. 1985 January; 10 p. (Format: TXT=22340 bytes)

[RFC-933] Silverman, S. Output marking Telnet option. 1985 January; 4 p. (Format: TXT=6943 bytes)

[RFC-932] Clark, D.D. Subnetwork addressing scheme. 1985 January; 4 p. (Format: TXT=9509 bytes)

[RFC-931] St. Johns, M. Authentication server. 1985 January; 4 p. (Format: TXT=9259 bytes) (Obsoletes RFC 912)

[RFC-930] Solomon, M.; Wimmers, E. Telnet terminal type option. 1985 January; 4 p. (Format: TXT=6805 bytes) (Obsoletes RFC 884; Obsoleted by RFC 1091)

[RFC-929] Lilienkamp, J.; Mandell, R.; Padlipsky, M.A. Proposed Host-Front End Protocol. 1984 December; 52 p. (Format: TXT=138234 bytes)

[RFC-928] Padlipsky, M.A. Introduction to proposed DoD standard H-FP. 1984 December; 21 p. (Format: TXT=61658 bytes)

[RFC-927] Anderson, B.A. TACACS user identification Telnet option. 1984 December; 4 p. (Format: TXT=5702 bytes)

[RFC-926] International Organization for Standardization. Protocol for providing the connectionless mode network services. 1984 December; 101 p. (Format: TXT=172024 bytes) (Obsoleted by RFC 994)

[RFC-925] Postel, J.B. Multi-LAN address resolution. 1984 October; 15 p. (Format: TXT=31992 bytes)

[RFC-924] Reynolds, J.K.; Postel, J.B. Official ARPA-Internet protocols for connecting personal computers to the Internet. 1984 October; 35 p. (Format: TXT=50543 bytes) (Obsoletes RFC 901; Obsoleted by RFC 944)

[RFC-923] Reynolds, J.K.; Postel, J.B. Assigned numbers. 1984 October; 47 p. (Format: TXT=99193 bytes) (Obsoletes RFC 900; Obsoleted by RFC 943)

[RFC-922] Mogul, J.C. Broadcasting Internet datagrams in the presence of subnets. 1984 October; 12 p. (Format: TXT=24832 bytes)

[RFC-921] Postel, J.B. Domain name system implementation schedule - revised. 1984 October; 13 p. (Format: TXT=24059 bytes) (Updates RFC 897)

[RFC-920] Postel, J.B.; Reynolds, J.K. Domain requirements. 1984 October; 14 p. (Format: TXT=28621 bytes)

[RFC-919] Mogul, J.C. Broadcasting Internet datagrams. 1984 October; 8 p. (Format: TXT=16838 bytes)

[RFC-918] Reynolds, J.K. Post Office Protocol. 1984 October; 5 p. (Format: TXT=10166 bytes) (Obsoleted by RFC 937)

[RFC-917] Mogul, J.C. Internet subnets. 1984 October; 22 p. (Format: TXT=48326 bytes)

[RFC-916] Finn, G.G. Reliable Asynchronous Transfer Protocol (RATP). 1984 October; 54 p. (Format: TXT=113815 bytes)

[RFC-915] Elvy, M.A.; Nedved, R. Network mail path service. 1984 December; 11 p. (Format: TXT=22262 bytes)

[RFC-914] Farber, D.J.; Delp, G.; Conte, T.M. Thinwire protocol for connecting personal computers to the Internet. 1984 September; 22 p. (Format: TXT=58586 bytes)

[RFC-913] Lottor, M. Simple File Transfer Protocol. 1984 September; 15 p. (Format: TXT=21784 bytes)

[RFC-912] St. Johns, M. Authentication service. 1984 September; 3 p. (Format: TXT=4715 bytes) (Obsoleted by RFC 931)

[RFC-911] Kirton, P. EGP Gateway under Berkeley UNIX 4.2. 1984 August 22; 22 p. (Format: TXT=57043 bytes)

[RFC-910] Forsdick, H.C. Multimedia mail meeting notes. 1984 August; 11 p. (Format: TXT=25553 bytes)

[RFC-909] Welles, C.; Milliken, W. Loader Debugger Protocol. 1984 July; 127 p. (Format: TXT=217583 bytes)

[RFC-908] Velten, D.; Hinden, R.M.; Sax, J. Reliable Data Protocol. 1984 July; 56 p. (Format: TXT=101185 bytes) (Updated by RFC 1151)

[RFC-907] Bolt Beranek and Newman, Inc. Host Access Protocol specification. 1984 July; 75 p. (Format: TXT=134566 bytes)

[RFC-906] Finlayson, R. Bootstrap loading using TFTP. 1984 June; 4 p. (Format: TXT=10329 bytes)

[RFC-905] McKenzie, A.M. ISO Transport Protocol specification ISO DP 8073. 1984 April; 154 p. (Format: TXT=258729 bytes) (Obsoletes RFC 892)

[RFC-904] Mills, D.L. Exterior Gateway Protocol formal specification. 1984 April; 30 p. (Format: TXT=65226 bytes) (Updates RFC 827, RFC 888)

[RFC-903] Finlayson, R.; Mann, T.; Mogul, J.C.; Theimer, M. Reverse Address Resolution Protocol. 1984 June; 4 p. (Format: TXT=9572 bytes)

[RFC-902] Reynolds, J.K.; Postel, J.B. ARPA Internet Protocol policy. 1984 July; 5 p. (Format: TXT=11317 bytes)

[RFC-901] Reynolds, J.K.; Postel, J.B. Official ARPA-Internet protocols. 1984 June; 28 p. (Format: TXT=42682 bytes) (Obsoletes RFC 880; Obsoleted by RFC 924)

[RFC-900] Reynolds, J.K.; Postel, J.B. Assigned Numbers. 1984 June; 43 p. (Format: TXT=84610 bytes) (Obsoletes RFC 870; Obsoleted by RFC 923)

[RFC-899] Postel, J.B.; Westine, A. Request For Comments summary notes: 800-899. 1984 May; 18 p. (Format: TXT=41028 bytes)

[RFC-898] Hinden, R.M.; Postel, J.B.; Muuss, M.; Reynolds, J.K. Gateway special interest group meeting notes. 1984 April; 24 p. (Format: TXT=43504 bytes)

[RFC-897] Postel, J.B. Domain name system implementation schedule. 1984 February; 8 p. (Format: TXT=16139 bytes) (Updates RFC 881; Updated by RFC 921)

[RFC-896] Nagle, J. Congestion control in IP/TCP internetworks. 1984 January 6; 9 p. (Format: TXT=27294 bytes)

[RFC-895] Postel, J.B. Standard for the transmission of IP datagrams over experimental Ethernet networks. 1984 April; 3 p. (Format: TXT=5156 bytes)

[RFC-894] Hornig, C. Standard for the transmission of IP datagrams over Ethernet networks. 1984 April; 3 p. (Format: TXT=5868 bytes)

[RFC-893] Leffler, S.; Karels, M.J. Trailer encapsulations. 1984 April; 3 p. (Format: TXT=13695 bytes)

[RFC-892] International Organization for Standardization. ISO Transport Protocol specification [Draft]. 1983 December; 82 p. (Format: TXT=162564 bytes) (Obsoleted by RFC 905)

[RFC-891] Mills, D.L. DCN local-network protocols. 1983 December; 26 p. (Format: TXT=66769 bytes)

[RFC-890] Postel, J.B. Exterior Gateway Protocol implementation schedule. 1984 February; 3 p. (Format: TXT=6070 bytes)

[RFC-889] Mills, D.L. Internet delay experiments. 1983 December; 12 p. (Format: TXT=27812 bytes)

[RFC-888] Seamonson, L.; Rosen, E.C. "STUB" Exterior Gateway Protocol. 1984 January; 38 p. (Format: TXT=55585 bytes) (Updated by RFC 904)

[RFC-887] Accetta, M. Resource Location Protocol. 1983 December; 16 p. (Format: TXT=37683 bytes)

[RFC-886] Rose, M.T. Proposed standard for message header munging. 1983 December 15; 16 p. (Format: TXT=31546 bytes)

[RFC-885] Postel, J.B. Telnet end of record option. 1983 December; 2 p. (Format: TXT=3346 bytes)

[RFC-884] Solomon, M.; Wimmers, E. Telnet terminal type option. 1983 December; 5 p. (Format: TXT=8166 bytes) (Obsoleted by RFC 930)

[RFC-883] Mockapetris, P.V. Domain names: Implementation specification. 1983 November; 73 p. (Format: TXT=179416 bytes) (Obsoleted by RFC 1034, RFC 1035; Updated by RFC 973)

[RFC-882] Mockapetris, P.V. Domain names: Concepts and facilities. 1983 November; 31 p. (Format: TXT=81574 bytes) (Obsoleted by RFC 1034, RFC 1035; Updated by RFC 973)

[RFC-881] Postel, J.B. Domain names plan and schedule. 1983 November; 10 p. (Format: TXT=24070 bytes) (Updated by RFC 897)

[RFC-880] Reynolds, J.K.; Postel, J.B. Official protocols. 1983 October; 26 p. (Format: TXT=38840 bytes) (Obsoletes RFC 840; Obsoleted by RFC 901)

[RFC-879] Postel, J.B. TCP maximum segment size and related topics. 1983 November; 11 p. (Format: TXT=22662 bytes)

[RFC-878] Malis, A.G. ARPANET 1822L Host Access Protocol. 1983 December; 48 p. (Format: TXT=77784 bytes) (Obsoletes RFC 851)

[RFC-877] Korb, J.T. Standard for the transmission of IP datagrams over public data networks. 1983 September; 2 p. (Format: TXT=3385 bytes)

[RFC-876] Smallberg, D. Survey of SMTP implementations. 1983 September; 13 p. (Format: TXT=38529 bytes)

[RFC-875] Padlipsky, M.A. Gateways, architectures, and heffalumps. 1982 September; 8 p. (Format: TXT=23380 bytes)

[RFC-874] Padlipsky, M.A. Critique of X.25. 1982 September; 13 p. (Format: TXT=37259 bytes)

[RFC-873] Padlipsky, M.A. Illusion of vendor support. 1982 September; 8 p. (Format: TXT=23673 bytes)

[RFC-872] Padlipsky, M.A. TCP-on-a-LAN. 1982 September; 8 p. (Format: TXT=22994 bytes)

[RFC-871] Padlipsky, M.A. Perspective on the ARPANET reference model. 1982 September; 25 p. (Format: TXT=76037 bytes)

[RFC-870] Reynolds, J.K.; Postel, J.B. Assigned numbers. 1983 October; 26 p. (Format: TXT=57563 bytes) (Obsoletes RFC 820; Obsoleted by RFC 900)

[RFC-869] Hinden, R.M. Host Monitoring Protocol. 1983 December; 70 p. (Format: TXT=98720 bytes)

[RFC-868] Postel, J.B.; Harrenstien, K. Time Protocol. 1983 May; 2 p. (Format: TXT=3140 bytes)

[RFC-867] Postel, J.B. Daytime Protocol. 1983 May; 2 p. (Format: TXT=2405 bytes)

[RFC-866] Postel, J.B. Active users. 1983 May; 1 p. (Format: TXT=2087 bytes)

[RFC-865] Postel, J.B. Quote of the Day Protocol. 1983 May; 1 p. (Format: TXT=1734 bytes)

[RFC-864] Postel, J.B. Character Generator Protocol. 1983 May; 3 p. (Format: TXT=7016 bytes)

[RFC-863] Postel, J.B. Discard Protocol. 1983 May; 1 p. (Format: TXT=1297 bytes)

[RFC-862] Postel, J.B. Echo Protocol. 1983 May; 1 p. (Format: TXT=1294 bytes)

[RFC-861] Postel, J.B.; Reynolds, J.K. Telnet extended options: List option. 1983 May; 1 p. (Format: TXT=3181 bytes) (Obsoletes NIC 16239)

[RFC-860] Postel, J.B.; Reynolds, J.K. Telnet timing mark option. 1983 May; 4 p. (Format: TXT=8108 bytes) (Obsoletes NIC 16238)

[RFC-859] Postel, J.B.; Reynolds, J.K. Telnet status option. 1983 May; 3 p. (Format: TXT=4443 bytes) (Obsoletes RFC 651)

[RFC-858] Postel, J.B.; Reynolds, J.K. Telnet Suppress Go Ahead option. 1983 May; 3 p. (Format: TXT=3825 bytes) (Obsoletes NIC 15392)

[RFC-857] Postel, J.B.; Reynolds, J.K. Telnet echo option. 1983 May; 5 p. (Format: TXT=11143 bytes) (Obsoletes NIC 15390)

[RFC-856] Postel, J.B.; Reynolds, J.K. Telnet binary transmission. 1983 May; 4 p. (Format: TXT=9192 bytes) (Obsoletes NIC 15389)

[RFC-855] Postel, J.B.; Reynolds, J.K. Telnet option specifications. 1983 May; 4 p. (Format: TXT=6218 bytes) (Obsoletes NIC 18640)

[RFC-854] Postel, J.B.; Reynolds, J.K. Telnet Protocol specification. 1983 May; 15 p. (Format: TXT=39371 bytes) (Obsoletes RFC 764, NIC 18639)

[RFC-853] Not issued.

[RFC-852] Malis, A.G. ARPANET short blocking feature. 1983 April; 13 p. (Format: TXT=17151 bytes)

[RFC-851] Malis, A.G. ARPANET 1822L Host Access Protocol. 1983 April 18; 44 p. (Format: TXT=72042 bytes) (Obsoletes RFC 802; Obsoleted by RFC 878)

[RFC-850] Horton, M.R. Standard for interchange of USENET messages. 1983 June; 18 p. (Format: TXT=43871 bytes) (Obsoleted by RFC 1036)

[RFC-849] Crispin, M. Suggestions for improved host table distribution. 1983 May; 2 p. (Format: TXT=5290 bytes)

[RFC-848] Smallberg, D. Who provides the "little" TCP services?. 1983 March 14; 5 p. (Format: TXT=11280 bytes)

[RFC-847] Smallberg, D.; Westine, A.; Postel, J.B. Summary of Smallberg surveys. 1983 February; 2 p. (Format: TXT=3906 bytes) (Obsoletes RFC 846)

[RFC-846] Smallberg, D. Who talks TCP? - survey of 22 February 1983. 1983 February 23; 14 p. (Format: TXT=46421 bytes) (Obsoletes RFC 845; Obsoleted by RFC 847)

[RFC-845] Smallberg, D. Who talks TCP? - survey of 15 February 1983. 1983 February 17; 14 p. (Format: TXT=46806 bytes) (Obsoletes RFC 843; Obsoleted by RFC 846)

[RFC-844] Clements, R. Who talks ICMP, too? - Survey of 18 February 1983. 1983 February 18; 5 p. (Format: TXT=9323 bytes) (Updates RFC 843)

[RFC-843] Smallberg, D. Who talks TCP? - survey of 8 February 83. 1983 February 9; 14 p. (Format: TXT=47023 bytes) (Obsoletes RFC 842; Obsoleted by RFC 845; Updated by RFC 844)

[RFC-842] Smallberg, D. Who talks TCP? - survey of 1 February 83. 1983 February 3; 14 p. (Format: TXT=46784 bytes) (Obsoletes RFC 839; Obsoleted by RFC 843)

[RFC-841] National Bureau of Standards. Specification for message format for Computer Based Message Systems. 1983 January 27; 110 p. (Format: TXT=238774 bytes) (Obsoletes RFC 806)

[RFC-840] Postel, J.B. Official protocols. 1983 April 13; 23 p. (Format: TXT=34868 bytes) (Obsoleted by RFC 880)

[RFC-839] Smallberg, D. Who talks TCP?. 1983 January 26; 14 p. (Format: TXT=45987 bytes) (Obsoletes RFC 838; Obsoleted by RFC 842)

[RFC-838] Smallberg, D. Who talks TCP?. 1983 January 20; 14 p. (Format: TXT=45844 bytes) (Obsoletes RFC 837; Obsoleted by RFC 839)

[RFC-837] Smallberg, D. Who talks TCP?. 1983 January 12; 14 p. (Format: TXT=45627 bytes) (Obsoletes RFC 836; Obsoleted by RFC 838)

[RFC-836] Smallberg, D. Who talks TCP?. 1983 January 5; 13 p. (Format: TXT=44397 bytes) (Obsoletes RFC 835; Obsoleted by RFC 837)

[RFC-835] Smallberg, D. Who talks TCP?. 1982 December 29; 13 p. (Format: TXT=43713 bytes) (Obsoletes RFC 834; Obsoleted by RFC 836)

[RFC-834] Smallberg, D. Who talks TCP?. 1982 December 22; 13 p. (Format: TXT=43512 bytes) (Obsoletes RFC 833; Obsoleted by RFC 835)

[RFC-833] Smallberg, D. Who talks TCP?. 1982 December 14; 13 p. (Format: TXT=43728 bytes) (Obsoletes RFC 832; Obsoleted by RFC 834)

[RFC-832] Smallberg, D. Who talks TCP?. 1982 December 7; 13 p. (Format: TXT=43518 bytes) (Obsoleted by RFC 833)

[RFC-831] Braden, R.T. Backup access to the European side of SATNET. 1982 December; 5 p. (Format: TXT=12090 bytes)

[RFC-830] Su, Z. Distributed system for Internet name service. 1982 October; 16 p. (Format: TXT=32585 bytes)

[RFC-829] Cerf, V.G. Packet satellite technology reference sources. 1982 November; 5 p. (Format: TXT=10919 bytes)

[RFC-828] Owen, K. Data communications: IFIP's international "network" of experts. 1982 August; 11 p. (Format: TXT=29922 bytes)

[RFC-827] Rosen, E.C. Exterior Gateway Protocol (EGP). 1982 October; 44 p. (Format: TXT=68436 bytes) (Updated by RFC 904)

[RFC-826] Plummer, D.C. Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48.bit Ethernet address for transmission on Ethernet hardware. 1982 November; 10 p. (Format: TXT=22026 bytes)

[RFC-825] Postel, J.B. Request for comments on Requests For Comments. 1982 November; 2 p. (Format: TXT=4255 bytes) (Obsoleted by RFC 1111)

[RFC-824] MacGregor, W.I.; Tappan D.C. CRONUS Virtual Local Network. 1982 August 25; 41 p. (Format: TXT=58732 bytes)

[RFC-823] Hinden, R.M.; Sheltzer, A. DARPA Internet gateway. 1982 September; 33 p. (Format: TXT=62620 bytes) (Updates IEN 109, IEN 30)

[RFC-822] Crocker, D. Standard for the format of ARPA Internet text messages. 1982 August 13; 47 p. (Format: TXT=109200 bytes) (Obsoletes RFC 733; Updated by RFC 1138, RFC 1148)

[RFC-821] Postel, J.B. Simple Mail Transfer Protocol. 1982 August; 58 p. (Format: TXT=124482 bytes) (Obsoletes RFC 788)

[RFC-820] Postel, J.B. Assigned numbers. 1982 August 14; 1 p. (Format: TXT=54213 bytes) (Obsoletes RFC 790; Obsoleted by RFC 870)

[RFC-819] Su, Z.; Postel, J.B. Domain naming convention for Internet user applications. 1982 August; 18 p. (Format: TXT=36358 bytes)

[RFC-818] Postel, J.B. Remote User Telnet service. 1982 November; 2 p. (Format: TXT=3809 bytes)

[RFC-817] Clark, D.D. Modularity and efficiency in protocol implementation. 1982 July; 26 p. (Format: TXT=47319 bytes)

[RFC-816] Clark, D.D. Fault isolation and recovery. 1982 July; 12 p. (Format: TXT=20754 bytes)

[RFC-815] Clark, D.D. IP datagram reassembly algorithms. 1982 July; 9 p. (Format: TXT=15028 bytes)

[RFC-814] Clark, D.D. Name, addresses, ports, and routes. 1982 July; 14 p. (Format: TXT=25426 bytes)

[RFC-813] Clark, D.D. Window and acknowlegement strategy in TCP. 1982 July; 22 p. (Format: TXT=39277 bytes)

[RFC-812] Harrenstien, K.; White, V. NICNAME/WHOIS. 1982 March 1; 3 p. (Format: TXT=5562 bytes) (Obsoleted by RFC 954)

[RFC-811] Harrenstien, K.; White, V.; Feinler, E.J. Hostnames Server. 1982 March 1; 5 p. (Format: TXT=8007 bytes) (Obsoleted by RFC 953)

[RFC-810] Feinler, E.J.; Harrenstien, K.; Su, Z.; White, V. DoD Internet host table specification. 1982 March 1; 9 p. (Format: TXT=14659 bytes) (Obsoletes RFC 608; Obsoleted by RFC 952)

[RFC-809] Chang, T. UCL facsimile system. 1982 February; 96 p. (Format: TXT=171153 bytes)

[RFC-808] Postel, J.B. Summary of computer mail services meeting held at BBN on 10 January 1979. 1982 March 1; 8 p. (Format: TXT=15930 bytes)

[RFC-807] Postel, J.B. Multimedia mail meeting notes. 1982 February 9; 6 p. (Format: TXT=11633 bytes)

[RFC-806] National Bureau of Standards. Proposed Federal Information Processing Standard: Specification for message format for computer based message systems. 1981 September; 99 p. (Format: TXT=216377 bytes) (Obsoleted by RFC 841)

[RFC-805] Postel, J.B. Computer mail meeting notes. 1982 February 8; 6 p. (Format: TXT=12522 bytes)

[RFC-804] International Telecommunication Union, International Telegraph and Telephone Consultative Committee. CCITT draft recommendation T.4 [Standardization of Group 3 facsimile apparatus for document transmission] 1981; 12 p. (Format: TXT=17025 bytes)

[RFC-803] Agarwal, A.; O'Connor, M.J.; Mills, D.L. Dacom 450/500 facsimile data transcoding. 1981 November 2; 14 p. (Format: TXT=33826 bytes)

[RFC-802] Malis, A.G. ARPANET 1822L Host Access Protocol. 1981 November; 43 p. (Format: TXT=62470 bytes) (Obsoleted by RFC 851)

[RFC-801] Postel, J.B. NCP/TCP transition plan. 1981 November; 21 p. (Format: TXT=42041 bytes)

[RFC-800] Postel, J.B.; Vernon, J. Request For Comments summary notes: 700-799. 1982 November; 10 p. (Format: TXT=18354 bytes)

[RFC-799] Mills, D.L. Internet name domains. 1981 September; 6 p. (Format: TXT=14189 bytes)

[RFC-798] Katz, A.R. Decoding facsimile data from the Rapicom 450. 1981 September; 17 p. (Format: TXT=39853 bytes)

[RFC-797] Katz, A.R. Format for Bitmap files. 1981 September; 2 p. (Format: TXT=3183 bytes)

[RFC-796] Postel, J.B. Address mappings. 1981 September; 7 p. (Format: TXT=11645 bytes) (Obsoletes IEN 115)

[RFC-795] Postel, J.B. Service mappings. 1981 September; 7 p. (Format: TXT=5460 bytes)

[RFC-794] Cerf, V.G. Pre-emption. 1981 September; 4 p. (Format: TXT=6022 bytes) (Updates IEN 125)

[RFC-793] Postel, J.B. Transmission Control Protocol. 1981 September; 85 p. (Format: TXT=177957 bytes)

[RFC-792] Postel, J.B. Internet Control Message Protocol. 1981 September; 21 p. (Format: TXT=30404 bytes) (Obsoletes RFC 777; Updated by RFC 950)

[RFC-791] Postel, J.B. Internet Protocol. 1981 September; 45 p. (Format: TXT=97779 bytes) (Obsoletes RFC 760)

[RFC-790] Postel, J.B. Assigned numbers. 1981 September; 15 p. (Format: TXT=36186 bytes) (Obsoletes RFC 776; Obsoleted by RFC 820)

[RFC-789] Rosen, E.C. Vulnerabilities of network control protocols: An example. 1981 July; 15 p. (Format: TXT=26440 bytes)

[RFC-788] Postel, J.B. Simple Mail Transfer Protocol. 1981 November; 62 p. (Format: TXT=112698 bytes) (Obsoletes RFC 780; Obsoleted by RFC 821)

[RFC-787] Chapin, A.L. Connectionless data transmission survey/tutorial. 1981 July; 41 p. (Format: TXT=86362 bytes)

[RFC-786] Sluizer, S.; Postel, J.B. Mail Transfer Protocol: ISI TOPS20 MTP-NIMAIL interface. 1981 July; 2 p. (Format: TXT=3245 bytes)

[RFC-785] Sluizer, S.; Postel, J.B. Mail Transfer Protocol: ISI TOPS20 file definitions. 1981 July; 3 p. (Format: TXT=7206 bytes)

[RFC-784] Sluizer, S.; Postel, J.B. Mail Transfer Protocol: ISI TOPS20 implementation. 1981 July; 3 p. (Format: TXT=6030 bytes)

[RFC-783] Sollins, K.R. TFTP Protocol (revision 2). 1981 June; 18 p. (Format: TXT=23522 bytes) (Obsoletes IEN 133)

[RFC-782] Nabielsky, J.; Skelton, A.P. Virtual Terminal management model. 1981; 20 p. (Format: TXT=44887 bytes)

[RFC-781] Su, Z. Specification of the Internet Protocol (IP) timestamp option. 1981 May; 2 p. (Format: TXT=4100 bytes)

[RFC-780] Sluizer, S.; Postel, J.B. Mail Transfer Protocol. 1981 May; 43 p. (Format: TXT=82951 bytes) (Obsoletes RFC 772; Obsoleted by RFC 788)

[RFC-779] Killian, E. Telnet send-location option. 1981 April; 2 p. (Format: TXT=2680 bytes)

[RFC-778] Mills, D.L. DCNET Internet Clock Service. 1981 April 18; 5 p. (Format: TXT=9689 bytes)

[RFC-777] Postel, J.B. Internet Control Message Protocol. 1981 April; 14 p. (Format: TXT=80232 bytes) (Obsoletes RFC 760; Obsoleted by RFC 792)

[RFC-776] Postel, J.B. Assigned numbers. 1981 January; 13 p. (Format: TXT=31065 bytes) (Obsoletes RFC 770; Obsoleted by RFC 790)

[RFC-775] Mankins, D.; Franklin, D.; Owen, A.D. Directory oriented FTP commands. 1980 December; 6 p. (Format: TXT=9822 bytes)

[RFC-774] Postel, J.B. Internet Protocol Handbook: Table of contents. 1980 October; 3 p. (Format: TXT=3625 bytes) (Obsoletes RFC 766)

[RFC-773] Cerf, V.G. Comments on NCP/TCP mail service transition strategy. 1980 October; 11 p. (Format: TXT=22818 bytes)

[RFC-772] Sluizer, S.; Postel, J.B. Mail Transfer Protocol. 1980 September; 31 p. (Format: TXT=62858 bytes) (Obsoleted by RFC 780)

[RFC-771] Cerf, V.G.; Postel, J.B. Mail transition plan. 1980 September; 9 p. (Format: TXT=19154 bytes)

[RFC-770] Postel, J.B. Assigned numbers. 1980 September; 15 p. (Format: TXT=27117 bytes) (Obsoletes RFC 762; Obsoleted by RFC 776)

[RFC-769] Postel, J.B. Rapicom 450 facsimile file format. 1980 September 26; 2 p. (Format: TXT=4194 bytes)

[RFC-768] Postel, J.B. User Datagram Protocol. 1980 August 28; 3 p. (Format: TXT=6069 bytes)

[RFC-767] Postel, J.B. Structured format for transmission of multi-media documents. 1980 August; 33 p. (Format: TXT=62316 bytes)

[RFC-766] Postel, J.B. Internet Protocol Handbook: Table of contents. 1980 July; 1 p. (Format: TXT=3585 bytes) (Obsoleted by RFC 774)

[RFC-765] Postel, J.B. File Transfer Protocol specification. 1980 June; 70 p. (Format: TXT=150771 bytes) (Obsoletes RFC 542; Obsoleted by RFC 959)

[RFC-764] Postel, J.B. Telnet Protocol specification. 1980 June; 15 p. (Format: TXT=40874 bytes) (Obsoleted by RFC 854)

[RFC-763] Abrams, M.D. Role mailboxes. 1980 May 7; 1 p. (Format: TXT=965 bytes)

[RFC-762] Postel, J.B. Assigned numbers. 1980 January; 13 p. (Format: TXT=25421 bytes) (Obsoletes RFC 758; Obsoleted by RFC 770)

[RFC-761] Postel, J.B. DoD standard Transmission Control Protocol. 1980 January; 84 p. (Format: TXT=172234 bytes)

[RFC-760] Postel, J.B. DoD standard Internet Protocol. 1980 January; 41 p. (Format: TXT=84214 bytes) (Obsoletes IEN 123; Obsoleted by RFC 791, RFC 777)

[RFC-759] Postel, J.B. Internet Message Protocol. 1980 August; 71 p. (Format: TXT=127948 bytes)

[RFC-758] Postel, J.B. Assigned numbers. 1979 August; 12 p. (Format: TXT=23606 bytes) (Obsoletes RFC 755; Obsoleted by RFC 762)

[RFC-757] Deutsch, D.P. Suggested solution to the naming, addressing, and delivery problem for ARPANET message systems. 1979 September 10; 17 p. (Format: TXT=36773 bytes)

[RFC-756] Pickens, J.R.; Feinler, E.J.; Mathis, J.E. NIC name server - a datagram-based information utility. 1979 July; 11 p. (Format: TXT=24172 bytes)

[RFC-755] Postel, J.B. Assigned numbers. 1979 May 3; 12 p. (Format: TXT=22734 bytes) (Obsoletes RFC 750; Obsoleted by RFC 758)

[RFC-754] Postel, J.B. Out-of-net host addresses for mail. 1979 April 6; 10 p. (Format: TXT=19791 bytes)

[RFC-753] Postel, J.B. Internet Message Protocol. 1979 March; 62 p. (Format: TXT=97006 bytes)

[RFC-752] Crispin, M. Universal host table. 1979 January 2; 13 p. (Format: TXT=34560 bytes)

[RFC-751] Lebling, P.D. Survey of FTP mail and MLFL. 1978 December 10; 5 p. (Format: TXT=10363 bytes)

[RFC-750] Postel, J.B. Assigned numbers. 1978 September 26; 10 p. (Format: TXT=20686 bytes) (Obsoletes RFC 739; Obsoleted by RFC 755)

[RFC-749] Greenberg, B. Telnet SUPDUP-Output option. 1978 September 18; 4 p. (Format: TXT=9160 bytes)

[RFC-748] Crispin, M. Telnet randomly-lose option. 1978 April 1; 2 p. (Format: TXT=2858 bytes)

[RFC-747] Crispin, M. Recent extensions to the SUPDUP Protocol. 1978 March 21; 3 p. (Format: TXT=2928 bytes)

[RFC-746] Stallman, R. SUPDUP graphics extension. 1978 March 17; 15 p. (Format: TXT=31081 bytes)

[RFC-745] Beeler, M. JANUS interface specifications. 1978 March 30; 10 p. (Format: TXT=(22042 bytes)

[RFC-744] Sattley, J. MARS - a Message Archiving and Retrieval Service. 1978 January 8; 6 p. (Format: TXT=11337 bytes)

[RFC-743] Harrenstien, K. FTP extension: XRSQ/XRCP. 1977 December 30; 8 p. (Format: TXT=16720 bytes)

[RFC-742] Harrenstien, K. NAME/FINGER Protocol. 1977 December 30; 7 p. (Format: TXT=12733 bytes) (Obsoleted by RFC 1194)

[RFC-741] Cohen, D. Specifications for the Network Voice Protocol (NVP). 1977 November 22; 30 p. (Format: TXT=59582 bytes)

[RFC-740] Braden, R.T. NETRJS Protocol. 1977 November 22; 19 p. (Format: TXT=39953 bytes) (Obsoletes RFC 599)

[RFC-739] Postel, J.B. Assigned numbers. 1977 November 11; 11 p. (Format: TXT=16983 bytes) (Obsoletes RFC 604, RFC 503; Obsoleted by RFC 750)

[RFC-738] Harrenstien, K. Time server. 1977 October 31; 1 p. (Format: TXT=1909 bytes)

[RFC-737] Harrenstien, K. FTP extension: XSEN. 1977 October 31; 1 p. (Format: TXT=2185 bytes)

[RFC-736] Crispin, M. Telnet SUPDUP option. 1977 October 31; 2 p. (Format: TXT=3200 bytes)

[RFC-735] Crocker, D.; Gumpertz, R.H. Revised Telnet byte macro option. 1977 November 3; 5 p. (Format: TXT=10879 bytes) (Obsoletes RFC 729)

[RFC-734] Crispin, M. SUPDUP Protocol. 1977 October 7; 14 p. (Format: TXT=33920 bytes)

[RFC-733] Crocker, D.; Vittal, J.; Pogran, K.T.; Henderson, D.A. Standard for the format of ARPA network text messages. 1977 November 21; 38 p. (Format: TXT=75001 bytes) (Obsoletes RFC 724; Obsoleted by RFC 822)

[RFC-732] Day, J.D. Telnet Data Entry Terminal option 1977 September 12; 30 p. (Format: TXT=58929 bytes) (Obsoletes RFC 731; Updated by RFC 1043)

[RFC-731] Day, J.D. Telnet Data Entry Terminal option. 1977 June 27; 28 p. (Format: TXT=63300 bytes) (Obsoleted by RFC 732)

[RFC-730] Postel, J.B. Extensible field addressing. 1977 May 20; 5 p. (Format: TXT=9812 bytes)

[RFC-729] Crocker, D. Telnet byte macro option. 1977 May 13; 4 p. (Format: TXT=6695 bytes) (Obsoleted by RFC 735)

[RFC-728] Day, J.D. Minor pitfall in the Telnet Protocol. 1977 April 27; 1 p. (Format: TXT=2265 bytes)

[RFC-727] Crispin, M. Telnet logout option. 1977 April 27; 3 p. (Format: TXT=5850 bytes)

[RFC-726] Postel, J.B.; Crocker, D. Remote Controlled Transmission and Echoing Telnet option. 1977 March 8; 16 p. (Format: TXT=39594 bytes)

[RFC-725] Day, J.D.; Grossman, G.R. RJE protocol for a resource sharing network. 1977 March 1; 26 p. (Format: TXT=45604 bytes)

[RFC-724] Crocker, D.; Pogran, K.T.; Vittal, J.; Henderson, D.A. Proposed official standard for the format of ARPA Network messages. 1977 May 12; 33 p. (Format: TXT=77423 bytes) (Obsoleted by RFC 733)

[RFC-723] Not issued.

[RFC-722] Haverty, J. Thoughts on interactions in distributed services. 1976 September 16; 20 p. (Format: TXT=30278 bytes)

[RFC-721] Garlick, L.L. Out-of-band control signals in a Host-to-Host Protocol. 1976 September 1; 7 p. (Format: TXT=13978 bytes)

[RFC-720] Crocker, D. Address specification syntax for network mail. 1976 August 5; 4 p. (Format: TXT=6835 bytes)

[RFC-719] Postel, J.B. Discussion on RCTE. 1976 July 22; 2 p. (Format: TXT=4823 bytes)

[RFC-718] Postel, J.B. Comments on RCTE from the Tenex implementation experience. 1976 June 30; 2 p. (Format: TXT=3944 bytes)

[RFC-717] Postel, J.B. Assigned network numbers. 1976 July 1; 2 p. (Format: TXT=2430 bytes)

[RFC-716] Walden, D.C.; Levin, J. Interim revision to Appendix F of BBN 1822. 1976 May 24; 2 p. (Format: TXT=3451 bytes)

[RFC-715] Not issued.

[RFC-714] McKenzie, A.M. Host-Host Protocol for an ARPANET-type network (Not online) 1976 April 21; 43 p.

[RFC-713] Haverty, J. MSDTP-Message Services Data Transmission Protocol. 1976 April 6; 29 p. (Format: TXT=42452 bytes)

[RFC-712] Donnelley, J.E. Distributed Capability Computing System (DCCS) (Not online) 1976 February 5; 38 p.

[RFC-711] Not issued.

[RFC-710] Not issued.

[RFC-709] Not issued.

[RFC-708] White, J.E. Elements of a distributed programming system. 1976 January 28; 29 p. (Format: TXT=59595 bytes)

[RFC-707] White, J.E. High-level framework for network-based resource sharing. 1975 December 23; 27 p. (Format: TXT=58900 bytes)

[RFC-706] Postel, J.B. On the junk mail problem. 1975 November 8; 1 p. (Format: TXT=2131 bytes)

[RFC-705] Bryan, R.F. Front-end Protocol B6700 version. 1975 November 5; 40 p. (Format: TXT=73143 bytes)

[RFC-704] Santos, P.J. IMP/Host and Host/IMP Protocol change. 1975 September 15; 3 p. (Format: TXT=7676 bytes) (Obsoletes RFC 687)

[RFC-703] Dodds, D.W. July, 1975, survey of New-Protocol Telnet Servers (Not online) 1975 July 11; 2 p.

[RFC-702] Dodds, D.W. September, 1974, survey of New-Protocol Telnet servers (Not online) 1974 September 25; 2 p.

[RFC-701] Dodds, D.W. August, 1974, survey of New-Protocol Telnet servers. 1974 August; 2 p. (Format: TXT=3662 bytes)

[RFC-700] Mader, E.; Plummer, W.W.; Tomlinson, R.S. Protocol experiment. 1974 August; 6 p. (Format: TXT=14931 bytes)

[RFC-699] Postel, J.B.; Vernon, J. Request For Comments summary notes: 600-699. 1982 November; 9 p. (Format: TXT=15219 bytes)

[RFC-698] Mock, T. Telnet extended ASCII option. 1975 July 23; 4 p. (Format: TXT=5307 bytes)

[RFC-697] Lieb, J. CWD command of FTP (Not online) 1975 July 14; 2 p.

[RFC-696] Cerf, V.G. Comments on the IMP/Host and Host/IMP Protocol changes (Not online) 1975 July 13; 2 p.

[RFC-695] Krilanovich, M. Official change in Host-Host Protocol. 1975 July 5; 2 p. (Format: TXT=3527 bytes)

[RFC-694] Postel, J.B. Protocol information (Not online) 1975 June 18; 36 p.

[RFC-693] Not issued.

[RFC-692] Wolfe, S.M. Comments on IMP/Host Protocol changes (RFCs 687 and 690) (Not online) 1975 June 20; 2 p. (Updates RFC 690)

[RFC-691] Harvey, B. One more try on the FTP. 1975 May 28; 13 p. (Format: TXT=33535 bytes)

[RFC-690] Postel, J.B. Comments on the proposed Host/IMP Protocol changes (Not online) 1975 June 6; 4 p. (Updates RFC 687; Updated by RFC 692)

[RFC-689] Clements, R. Tenex NCP finite state machine for connections. 1975 May 23; 6 p. (Format: TXT=13378 bytes)

[RFC-688] Walden, D.C. Tentative schedule for the new Telnet implementation for the TIP (Not online) 1975 June 4; 1 p.

[RFC-687] Walden, D.C. IMP/Host and Host/IMP Protocol changes. 1975 June 2; 3 p. (Format: TXT=6183 bytes) (Obsoleted by RFC 704; Updated by RFC 690)

[RFC-686] Harvey, B. Leaving well enough alone (Not online) 1975 May 10; 9 p.

[RFC-685] Beeler, M. Response time in cross network debugging. 1975 April 16; 4 p. (Format: TXT=7084 bytes)

[RFC-684] Schantz, R. Commentary on procedure calling as a network protocol. 1975 April 15; 7 p. (Format: TXT=21575 bytes)

[RFC-683] Clements, R. FTPSRV - Tenex extension for paged files. 1975 April 3; 9 p. (Format: TXT=8981 bytes)

[RFC-682] Not issued.

[RFC-681] Holmgren, S. Network UNIX. 1975 March 18; 6 p. (Format: TXT=19305 bytes)

[RFC-680] Myer, T.H.; Henderson, D.A. Message Transmission Protocol (Not online) 1975 April 30; 6 p. (Updates RFC 561)

[RFC-679] Dodds, D.W. February, 1975, survey of New-Protocol Telnet servers (Not online) 1975 February 21; 2 p.

[RFC-678] Postel, J.B. Standard file formats. 1974 December 19; 8 p. (Format: TXT=12865 bytes)

[RFC-677] Johnson, P.R.; Thomas, R. Maintenance of duplicate databases (Not online) 1975 January 27; 9 p.

[RFC-676] Not issued.

[RFC-675] Cerf, V.G.; Dalal, Y.K.; Sunshine, C.A. Specification of Internet Transmission Control Program (Not online) 1974 December; 70 p.

[RFC-674] Postel, J.B.; White, J.E. Procedure call documents - version 2. 1974 December 12; 4 p. (Format: TXT=12475 bytes)

[RFC-673] Not issued.

[RFC-672] Schantz, R. Multi-site data collection facility. 1974 December 6; 10 p. (Format: TXT=26279 bytes)

[RFC-671] Schantz, R. Note on Reconnection Protocol (Not online) 1974 December 6; 8 p.

[RFC-670] Not issued.

[RFC-669] Dodds, D.W. November, 1974, survey of New-Protocol Telnet servers (Not online) 1974 December 4; 4 p.

[RFC-668] Not issued.

[RFC-667] Chipman, S.G. BBN host ports (Not online) 1974 December 17; 1 p.

[RFC-666] Padlipsky, M.A. Specification of the Unified User-Level Protocol (Not online) 1974 November 26; 17 p.

[RFC-665] Not issued.

[RFC-664] Not issued.

[RFC-663] Kanodia, R. Lost message detection and recovery protocol. 1974 November 29; 17 p. (Format: TXT=45956 bytes)

[RFC-662] Kanodia, R. Performance improvement in ARPANET file transfers from Multics. 1974 November 26; 3 p. (Format: TXT=9048 bytes)

[RFC-661] Postel, J.B. Protocol information (Not online) 1974 November 23; 23 p.

[RFC-660] Walden, D.C. Some changes to the IMP and the IMP/Host interface. 1974 October 23; 2 p. (Format: TXT=5106 bytes)

[RFC-659] Postel, J.B. Announcing additional Telnet options (Not online) 1974 October 18; 1 p.

[RFC-658] Crocker, D. Telnet output linefeed disposition. 1974 October 25; 4 p. (Format: TXT=6603 bytes)

[RFC-657] Crocker, D. Telnet output vertical tab disposition option. 1974 October 25; 4 p. (Format: TXT=5871 bytes)

[RFC-656] Crocker, D. Telnet output vertical tabstops option. 1974 October 25; 3 p. (Format: TXT=4952 bytes)

[RFC-655] Crocker, D. Telnet output formfeed disposition option. 1974 October 25; 4 p. (Format: TXT=6105 bytes)

[RFC-654] Crocker, D. Telnet output horizontal tab disposition option. 1974 October 25; 4 p. (Format: TXT=6270 bytes)

[RFC-653] Crocker, D. Telnet output horizontal tabstops option. 1974 October 25; 3 p. (Format: TXT=4782 bytes)

[RFC-652] Crocker, D. Telnet output carriage-return disposition option. 1974 October 25; 3 p. (Format: TXT=7165 bytes)

[RFC-651] Crocker, D. Revised Telnet status option. 1974 October 25; 3 p. (Format: TXT=4446 bytes) (Obsoleted by RFC 859)

RFC-1001 Protocol Standard for a NetBIOS Service on a TCP/UDP Transport: Concepts and Methods

March 1987

Abstract

This RFC defines an elective standard protocol to support NetBIOS services in a TCP/IP environment. Both local network and internet operation are supported. Various node types are defined to accommodate local and internet topologies and to allow operation with or without the use of IP broadcast.

This RFC describes the NetBIOS-over-TCP protocols in a general manner, emphasizing the underlying ideas and techniques. Detailed specifications are found in a companion RFC, "Protocol Standard For a NetBIOS Service on a TCP/UDP Transport: Detailed Specifications" [RFC-1002].

Design Principles

RFC-1001 Protocol Standard for a NetBIOS Service - Concepts and Methods

Overview of NetBIOS

RFC-1001 Protocol Standard for a NetBIOS Service - Concepts and Methods

NetBIOS Support Servers

Topologies

RFC-1001 Protocol Standard for a NetBIOS Service - Concepts and Methods

General Methods

RFC-1001 Protocol Standard for a NetBIOS Service - Concepts and Methods

NetBIOS Name Service

RFC-1001 Protocol Standard for a NetBIOS Service - Concepts and Methods

NetBIOS Session Service

RFC-1001 Protocol Standard for a NetBIOS Service - Concepts and Methods

NetBIOS Datagram Service

Status Of This Memo

This RFC specifies an Elective Standard Protocol for the Internet Community.

Please send written comments to:

Karl Auerbach
Epilogue Technology Corporation
P.O. Box 5432
Redwood City, CA 94063

Please send online comments to:

Avnish Aggarwal
Internet: mtxinu!excelan!avnish@ucbvax.berkeley.edu
Usenet: ucbvax!mtxinu!excelan!avnish

Acknowledgements

This RFC has been developed under the auspices of the Internet Activities Board, especially the End-to-End Services Task Force.

The following individuals have contributed to the development of this RFC:

Avnish Aggarwal	Arvind Agrawal	Lorenzo Aguilar
Geoffrey Arnold	Karl Auerbach	K. Ramesh Babu
Keith Ball	Amatzia Ben-Artzi	Vint Cerf
Richard Cherry	David Crocker	Steve Deering
Greg Ennis	Steve Holmgren	Jay Israel
David Kaufman	Lee LaBarre	James Lau
Dan Lynch	Gaylord Miyata	David Stevens
Steve Thomas	Ishan Wu	

The system proposed by this RFC does not reflect any existing Netbios-over-TCP implementation. However, the design incorporates considerable knowledge obtained from prior implementations. Special thanks goes to the following organizations which have provided this invaluable information:

CMC/Syros Excelan Sytek Ungermann-Bass

RFC-1001 Protocol Standard for a NetBIOS Service - Concepts and Methods

Introduction

This RFC describes the ideas and general methods used to provide NetBIOS on a TCP and UDP foundation. A companion RFC, "Protocol Standard For a NetBIOS Service on a TCP/UDP Transport: Detailed Specifications"[RFC-1002] contains detailed descriptions of packet formats, protocols, and defined constants and variables.

The NetBIOS service has become the dominant mechanism for personal computer networking. NetBIOS provides a vendor independent interface for the IBM Personal Computer (PC) and compatible systems.

NetBIOS defines a software interface not a protocol. There is no "official" NetBIOS service standard. In practice, however, the IBM PC-Network version is used as a reference. That version is described in the IBM document 6322916, "Technical Reference PC Network"[2].

Protocols supporting NetBIOS services have been constructed on diverse protocol and hardware foundations. Even when the same foundation is used, different implementations may not be able to interoperate unless they use a common protocol. To allow NetBIOS interoperation in the Internet, this RFC defines a standard protocol to support NetBIOS services using TCP and UDP.

NetBIOS has generally been confined to personal computers to date. However, since larger computers are often well suited to run certain NetBIOS applications, such as file servers, this specification has been designed to allow an implementation to be built on virtually any type of system where the TCP/IP protocol suite is available.

This standard defines a set of protocols to support NetBIOS services.

These protocols are more than a simple communications service involving two entities. Rather, this note describes a distributed system in which many entities play a part even if they are not involved as an end-point of a particular NetBIOS connection.

This standard neither constrains nor determines how those services are presented to application programs.

Nevertheless, it is expected that on computers operating under the PC-DOS and MS-DOS operating systems that the existing NetBIOS interface will be preserved by implementors.

NOTE: Various symbolic values are used in this document. For their definitions, refer to the Detailed Specifications[RFC-1002].

Design Principles

In order to develop the specification the following design principles were adopted to guide the effort. Most are typical to any protocol standardization effort; however, some have been assigned priorities that may be considered unusual.

Preserve NetBIOS Services

In the absence of an "official" standard for NetBIOS services, the version found in the IBM PC Network Technical Reference[2] is used.

NetBIOS is the foundation of a large body of existing applications. It is desirable to operate these applications on TCP networks and to extend them beyond personal computers into larger hosts. To support these applications, NetBIOS on TCP must closely conform to the services offered by existing NetBIOS systems.

IBM PC-Network NetBIOS contains some implementation specific characteristics. This standard does not attempt to completely preserve these. It is certain that some existing software requires these characteristics and will fail to operate correctly on a NetBIOS service based on this RFC.

Use Existing Standards

Protocol development, especially with standardization, is a demanding process. The development of new protocols must be minimized.

It is considered essential that an existing standard which provides the necessary functionality with reasonable performance always be chosen in preference to developing a new protocol.

When a standard protocol is used, it must be unmodified.

Minimize Options

The standard for NetBIOS on TCP should contain few, if any, options.

Where options are included, the options should be designed so that devices with different option selections should interoperate.

Tolerate Errors and Disruptions

NetBIOS networks typically operate in an uncontrolled environment. Computers come on-line at arbitrary times. Computers usually go off-line without any notice to their peers. The software is often operated by users who are unfamiliar with networks and who may randomly perturb configuration settings.

Despite this chaos, NetBIOS networks work. NetBIOS on TCP must also be able to operate well in this environment.

Robust operation does not necessarily mean that the network is proof against all disruptions. A typical NetBIOS network may be disrupted by certain types of behavior, whether inadvertent or malicious.

Do Not Require Central Management

NetBIOS on TCP should be able to operate, if desired, without centralized management beyond that typically required by a TCP based network.

Allow Internet Operation

The proposed standard recognizes the need for NetBIOS operation across a set of networks interconnected by network (IP) level relays (gateways.)

However, the standard assumes that this form of operation will be less frequent than on the local MAC bridged-LAN.

Minimize Broadcast Activity

The standard pre-supposes that the only broadcast services are those supported by UDP. Multicast capabilities are not assumed to be available in any form.

Despite the availability of broadcast capabilities, the standard recognizes that some administrations may wish to avoid heavy broadcast activity. For example, an administration may wish to avoid isolated non-participating hosts from the burden of receiving and discarding NetBIOS broadcasts.

Permit Implementation On Existing Systems

The NetBIOS on TCP protocol should be implementable on common operating systems, such as Unix(tm) and VAX/VMS(tm), without massive effort.

The NetBIOS protocols should not require services typically unavailable on presently existing TCP/UDP/IP implementations.

Require Only the Minimum Necessary to Operate

The protocol definition should specify only the minimal set of protocols required for interoperation. However, additional protocol elements may be defined to enhance efficiency. These latter elements may be generated at the option of the sender, although they must be accepted by all receivers.

Maximize Efficiency

To be useful, a protocol must conduct its business quickly.

Minimize New Inventions

When an existing protocol is not quite able to support a necessary function, but with a small amount of change, it could, that protocol should be used. This is felt to be easier to achieve than development of new protocols; further, it is likely to have more general utility for the Internet.

RFC-1001 Protocol Standard for a NetBIOS Service - Concepts and Methods

Overview of NetBIOS

This section describes the NetBIOS services. It is for background information only. The reader may chose to skip to the next section.

NetBIOS was designed for use by groups of PCs, sharing a broadcast medium. Both connection (Session) and connectionless (Datagram) services are provided, and broadcast and multicast are supported. Participants are identified by name. Assignment of names is distributed and highly dynamic.

NetBIOS applications employ NetBIOS mechanisms to locate resources, establish connections, send and receive data with an application peer, and terminate connections. For purposes of discussion, these mechanisms will collectively be called the NetBIOS Service.

This service can be implemented in many different ways. One of the first implementations was for personal computers running the PC-DOS and MS-DOS operating systems. It is possible to implement NetBIOS within other operating systems, or as processes which are, themselves, simply application programs as far as the host operating system is concerned.

The NetBIOS specification, published by IBM as "Technical Reference PC Network"[2] defines the interface and services available to the NetBIOS user. The protocols outlined by that document pertain only to the IBM PC Network and are not generally applicable to other networks.

Interface To Application Programs

NetBIOS on personal computers includes both a set of services and an exact program interface to those services. NetBIOS on other computer systems may present the NetBIOS services to programs using other interfaces. Except on personal computers, no clear standard for a NetBIOS software interface has emerged.

RFC-1001 Protocol Standard for a NetBIOS Service - Overview of NetBIOS

Name Service

NetBIOS resources are referenced by name. Lower-level address information is not available to NetBIOS applications. An application, representing a resource, registers one or more names that it wishes to use.

The name space is flat and uses sixteen alphanumeric characters. Names may not start with an asterisk (*).

Registration is a bid for use of a name. The bid may be for exclusive (unique) or shared (group) ownership. Each application contends with the other applications in real time. Implicit permission is granted to a station when it receives no objections. That is, a bid is made and the application waits for a period of time. If no objections are received, the station assumes that it has permission.

A unique name should be held by only one station at a time. However, duplicates ("name conflicts") may arise due to errors.

All instances of a group name are equivalent.

An application referencing a name generally does not know (or care) whether the name is registered as a unique or a group name.

An explicit name deletion function is specified, so that applications may remove a name. Implicit name deletion occurs when a station ceases operation. In the case of personal computers, implicit name deletion is a frequent occurrence.

The Name Service primitives are:

1) Add Name
   The requesting application wants exclusive use of the name.

2) Add Group Name
   The requesting application is willing to share use of the name with other applications.

3) Delete Name
   The application no longer requires use of the name. It is important to note that typical use of NetBIOS is among independently-operated personal computers. A common way to stop using a PC is to turn it off; in this case, the graceful give-back mechanism, provided by the Delete Name function, is not used. Because this occurs frequently, the network service must support this behavior.

RFC-1001 Protocol Standard for a NetBIOS Service - Overview of NetBIOS

Session Service

A session is a reliable message exchange, conducted between a pair of NetBIOS applications. Sessions are full-duplex, sequenced, and reliable. Data is organized into messages. Each message may range in size from 0 to 131,071 bytes. No expedited or urgent data capabilities are present.

Multiple sessions may exist between any pair of calling and called names.

The parties to a connection have access to the calling and called names.

The NetBIOS specification does not define how a connection request to a shared (group) name resolves into a session. The usual assumption is that a session may be established with any one owner of the called group name.

An important service provided to NetBIOS applications is the detection of sessions failure. The loss of a session is reported to an application via all of the outstanding service requests for that session. For example, if the application has only a NetBIOS receive primitive pending and the session terminates, the pending receive will abort with a termination indication.

Session Service primitives are:

1) Call
   Initiate a session with a process that is listening under the specified name. The calling entity must indicate both a calling name (properly registered to the caller) and a called name.

2) Listen
   Accept a session from a caller. The listen primitive may be constrained to accept an incoming call from a named caller. Alternatively, a call may be accepted from any caller.

3) Hang Up
   Gracefully terminate a session. All pending data is transferred before the session is terminated.

4) Send
   Transmit one message. A time-out can occur. A time-out of any session send forces the non-graceful termination of the session. A "chain send" primitive is required by the PC NetBIOS software interface to allow a single message to be gathered from pieces in various buffers. Chain Send is an interface detail and does not effect the protocol.

5) Receive
   Receive data. A time-out can occur. A time-out on a session receive only terminates the receive, not the session, although the data is lost.

   The receive primitive may be implemented with variants, such as "Receive Any", which is required by the PC NetBIOS software interface. Receive Any is an interface detail and does not effect the protocol.

6) Session Status
   Obtain information about all of the requestor's sessions, under the specified name. No network activity is involved.

Datagram Service

The Datagram service is an unreliable, non-sequenced, connectionless service. Datagrams are sent under cover of a name properly registered to the sender.

Datagrams may be sent to a specific name or may be explicitly broadcast.

Datagrams sent to an exclusive name are received, if at all, by the holder of that name. Datagrams sent to a group name are multicast to all holders of that name. The sending application program cannot distinguish between group and unique names and thus must act as if all non-broadcast datagrams are multicast.

As with the Session Service, the receiver of the datagram is told the sending and receiving names.

Datagram Service primitives are:

1) Send Datagram
   Send an unreliable datagram to an application that is associated with the specified name. The name may be unique or group; the sender is not aware of the difference. If the name belongs to a group, then each member is to receive the datagram.

2) Send Broadcast Datagram
   Send an unreliable datagram to any application with a Receive Broadcast Datagram posted.

3) Receive Datagram
   Receive a datagram sent by a specified originating name to the specified name. If the originating name is an asterisk, then the datagram may have been originated under any name.

   Note: An arriving datagram will be delivered to all pending Receiving Datagrams that have source and destination specifications matching those of the datagram. In other words, if a program (or group of programs) issue a series of identical Receive Datagrams, one datagram will cause the entire series to complete.

4) Receive Broadcast Datagram
   Receive a datagram sent as a broadcast.

   If there are multiple pending Receive Broadcast Datagram operations pending, all will be satisfied by the same received datagram.

Miscellaneous Functions

The following functions are present to control the operation of the hardware interface to the network. These functions are generally implementation dependent.

1) Reset
   Initialize the local network adapter.

2) Cancel
   Abort a pending NetBIOS request. The successful cancel of a Send (or Chain Send) operation will terminate the associated session.

3) Adapter Status
   Obtain information about the local network adapter or of a remote adapter.

4) Unlink
   For use with Remote Program Load (RPL). Unlink redirects the PC boot disk device back to the local disk. See the NetBIOS specification for further details concerning RPL and the Unlink operation (see page 2-35 in [2]).

5) Remote Program Load
   Remote Program Load (RPL) is not a NetBIOS function. It is a NetBIOS application defined by IBM in their NetBIOS specification (see pages 2-80 through 2-82 in [2]).

Non-Standard Extensions

The IBM Token Ring implementation of NetBIOS has added at least one new user capability:

1) Find Name
   This function determines whether a given name has been registered on the network.

NetBIOS Facilities Supported by this Standard

The protocol specified by this standard permits an implementer to provide all of the NetBIOS services as described in the IBM "Technical Reference PC Network"[2].

The following NetBIOS facilities are outside the scope of this specification. These are local implementation matters and do not impact interoperability:

- RESET
- SESSION STATUS
- UNLINK
- RPL (Remote Program Load)

Required Supporting Service Interfaces and Definitions

The protocols described in this RFC require service interfaces to the following:

- TCP [RFC-793]
- UDP [RFC-768]

Byte ordering, addressing conventions (including addresses to be used for broadcasts and multicasts) are defined by the most recent version of:

- Assigned Numbers [RFC-1060]

Additional definitions and constraints are in:

- IP [RFC-791]
- Internet Subnets: "Internet Subnets" [RFC-950]
  "Broadcasting IP Datagrams in the Presence of Subnets" [RFC-922]

Related Protocols And Services

The design of the protocols described in this RFC allow for the future incorporation of the following protocols and services. However, before this may occur, certain extensions may be required to the protocols defined in this RFC or to those listed below.

- Domain Name Service [11,12,13,14]
- Internet Group Multicast [15,16]

NetBIOS Scope

A "NetBIOS Scope" is the population of computers across which a registered NetBIOS name is known. NetBIOS broadcast and multicast datagram operations must reach the entire extent of the NetBIOS scope.

An internet may support multiple, non-intersecting NetBIOS Scopes.

Each NetBIOS scope has a "scope identifier". This identifier is a character string meeting the requirements of the domain name system for domain names.

NOTE: Each implementation of NetBIOS-over-TCP must provide mechanisms to manage the scope identifier(s) to be used.

Control of scope identifiers implies a requirement for additional NetBIOS interface capabilities. These may be provided through extensions of the user service interface or other means (such as node configuration parameters.) The nature of these extensions is not part of this specification.

NetBIOS End-Nodes

End-nodes support NetBIOS service interfaces and contain applications.

Three types of end-nodes are part of this standard:

- Broadcast ("B") nodes
- Point-to-point ("P") nodes
- Mixed mode ("M") nodes

An IP address may be associated with only one instance of one of the above types.

Without having preloaded name-to-address tables, NetBIOS participants are faced with the task of dynamically resolving references to one another. This can be accomplished with broadcast or mediated point- to-point communications.

B nodes use local network broadcasting to effect a rendezvous with one or more recipients. P and M nodes use the NetBIOS Name Server (NBNS) and the NetBIOS Datagram Distribution Server (NBDD) for this same purpose.

End-nodes may be combined in various topologies. No matter how combined, the operation of the B, P, and M nodes is not altered.

NOTE: It is recommended that the administration of a NetBIOS scope avoid using both M and B nodes within the same scope. A NetBIOS scope should contain only B nodes or only P and M nodes.

Broadcast (B) Nodes

Broadcast (or "B") nodes communicate using a mix of UDP datagrams (both broadcast and directed) and TCP connections. B nodes may freely interoperate with one another within a broadcast area. A broadcast area is a single MAC-bridged "B-LAN". (See Appendix A for a discussion of using Internet Group Multicasting as a means to extend a broadcast area beyond a single B-LAN.)

Point-To-Point (P) Nodes

Point-to-point (or "P") nodes communicate using only directed UDP datagrams and TCP sessions. P nodes neither generate nor listen for broadcast UDP packets. P nodes do, however, offer NetBIOS level broadcast and multicast services using capabilities provided by the NBNS and NBDD.

P nodes rely on NetBIOS name and datagram distribution servers. These servers may be local or remote; P nodes operate the same in either case.

Mixed Mode (M) Nodes

Mixed mode nodes (or "M") nodes are P nodes which have been given certain B node characteristics. M nodes use both broadcast and unicast. Broadcast is used to improve response time using the assumption that most resources reside on the local broadcast medium rather than somewhere in an internet.

M nodes rely upon NBNS and NBDD servers. However, M nodes may continue limited operation should these servers be temporarily unavailable.

NetBIOS Support Servers

Two types of support servers are part of this standard:

- NetBIOS name server ("NBNS") nodes
- NetBIOS datagram distribution ("NBDD") nodes

NBNS and NBDD nodes are invisible to NetBIOS applications and are part of the underlying NetBIOS mechanism.

NetBIOS name and datagram distribution servers are the focus of name and datagram activity for P and M nodes.

Both the name (NBNS) and datagram distribution (NBDD) servers are permitted to shift part of their operation to the P or M end-node which is requesting a service.

Since the assignment of responsibility is dynamic, and since P and M nodes must be prepared to operate should the NetBIOS server delegate control to the maximum extent, the system naturally accommodates improvements in NetBIOS server function. For example, as Internet Group Multicasting becomes more widespread, new NBDD implementations may elect to assume full responsibility for NetBIOS datagram distribution.

Interoperability between different implementations is assured by imposing requirements on end-node implementations that they be able to accept the full range of legal responses from the NBNS or NBDD.

Relationship of the NBNS to the Domain Name System
Relationship of NBNS and NBDD Nodes
Relationship of NetBIOS Support Servers and B Nodes

NetBIOS Name Server (NBNS) Nodes

The NBNS is designed to allow considerable flexibility with its degree of responsibility for the accuracy and management of NetBIOS names. On one hand, the NBNS may elect not to accept full responsibility, leaving the NBNS essentially a "bulletin board" on which name/address information is freely posted (and removed) by P and M nodes without validation by the NBNS. Alternatively, the NBNS may elect to completely manage and validate names. The degree of responsibility that the NBNS assumes is asserted by the NBNS each time a name is claimed through a simple mechanism. Should the NBNS not assert full control, the NBNS returns enough information to the requesting node so that the node may challenge any putative holder of the name.

This ability to shift responsibility for NetBIOS name management between the NBNS and the P and M nodes allows a network administrator (or vendor) to make a tradeoff between NBNS simplicity, security, and delay characteristics.

A single NBNS may be implemented as a distributed entity, such as the Domain Name Service. However, this RFC does not attempt to define the internal communications which would be used.

Relationship of the NBNS to the Domain Name System

The NBNS design attempts to align itself with the Domain Name System in a number of ways.

First, the NetBIOS names are encoded in a form acceptable to the domain name system.

Second, a scope identifier is appended to each NetBIOS name. This identifier meets the restricted character set of the domain system and has a leading period. This makes the NetBIOS name, in conjunction with its scope identifier, a valid domain system name.

Third, the negotiated responsibility mechanisms permit the NBNS to be used as a simple bulletin board on which are posted (name,address) pairs. This parallels the existing domain sytem query service.

This RFC, however, requires the NBNS to provide services beyond those provided by the current domain name system. An attempt has been made to coalesce all the additional services which are required into a set of transactions which follow domain name system styles of interaction and packet formats.

Among the areas in which the domain name service must be extended before it may be used as an NBNS are:

- Dynamic addition of entries
- Dynamic update of entry data
- Support for multiple instance (group) entries
- Support for entry time-to-live values and ability to accept refresh messages to restart the time-to-live period
- New entry attributes

NetBIOS Datagram Distribution Server (NBDD) Nodes

The internet does not yet support broadcasting or multicasting, though it is likely to in the near future. Several specifications are already in place to support both of these facilities; See "Broadcasting IP Datagrams in the Presence of Subnets" [RFC-922] and "Host Extensions for IP Multicasting" [RFC-1112] and "Internet Group Multicast Protocol". The NBDD extends NetBIOS datagram distribution service to this environment.

The NBDD may elect to complete, partially complete, or totally refuse to service a node's request to distribute a NetBIOS datagram. An end-node may query an NBDD to determine whether the NBDD will deliver a datagram to a specific NetBIOS name.

The design of NetBIOS-over-TCP lends itself to the use of Internet Group Multicast. For details see Appendix A.

Relationship of NBNS and NBDD Nodes

This RFC defines the NBNS and NBDD as distinct, separate entities.

In the absence of NetBIOS name information, a NetBIOS datagram distribution server must send a copy to each end-node within a NetBIOS scope.

An implementer may elect to construct NBNS and NBDD nodes which have a private protocol for the exchange of NetBIOS name information. Alternatively, an NBNS and NBDD may be implemented within the same device.

NOTE: Implementations containing private NBNS-NBDD protocols or combined NBNS-NBDD functions must be clearly identified.

Relationship of NetBIOS Support Servers and B Nodes

As defined in this RFC, neither NBNS nor NBDD nodes interact with B nodes. NetBIOS servers do not listen to broadcast traffic on any broadcast area to which they may be attached. Nor are the NetBIOS support servers even aware of B node activities or names claimed or used by B nodes.

It may be possible to extend both the NBNS and NBDD so that they participate in B node activities and act as a bridge to P and M nodes. However, such extensions are beyond the scope of this specification.

Topologies

B, P, M, NBNS, and NBDD nodes may be combined in various ways to form useful NetBIOS environments. This section describes some of these combinations.

There are three classes of operation:

- Class 0: B nodes only.
- Class 1: P nodes only.
- Class 2: P and M nodes together.

In the drawings which follow, any P node may be replaced by an M node. The effects of such replacement will be mentioned in conjunction with each example below.

Local

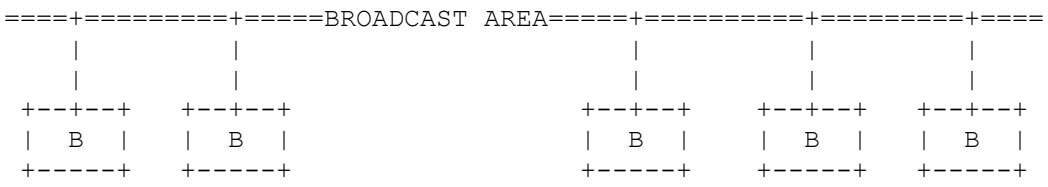
A NetBIOS scope is operating locally when all entities are within the same broadcast area.

B Nodes Only
P Nodes Only
Mixed B and P Nodes

RFC-1001 Protocol Standard for a NetBIOS Service - Local Topologies

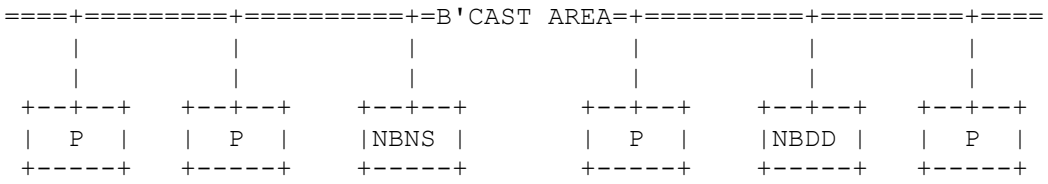
B Nodes Only

Local operation with only B nodes is the most basic mode of operation. Name registration and discovery procedures use broadcast mechanisms. The NetBIOS scope is limited by the extent of the broadcast area. This configuration does not require NetBIOS support servers.



P Nodes Only

This configuration would typically be used when the network administrator desires to eliminate NetBIOS as a source of broadcast activity.



This configuration operates the same as if it were in an internet and is cited here only due to its convenience as a means to reduce the use of broadcast.

Replacement of one or more of the P nodes with M nodes will not affect the operation of the other P and M nodes. P and M nodes will be able to interact with one another. Because M nodes use broadcast, overall broadcast activity will increase.

Mixed B and P Nodes

B and P nodes do not interact with one another. Replacement of P nodes with M nodes will allow B's and M's to interact.

NOTE: B nodes and M nodes may be intermixed only on a local broadcast area. B and M nodes should not be intermixed in an internet environment.

RFC-1001 Protocol Standard for a NetBIOS Service - Topologies

Internet

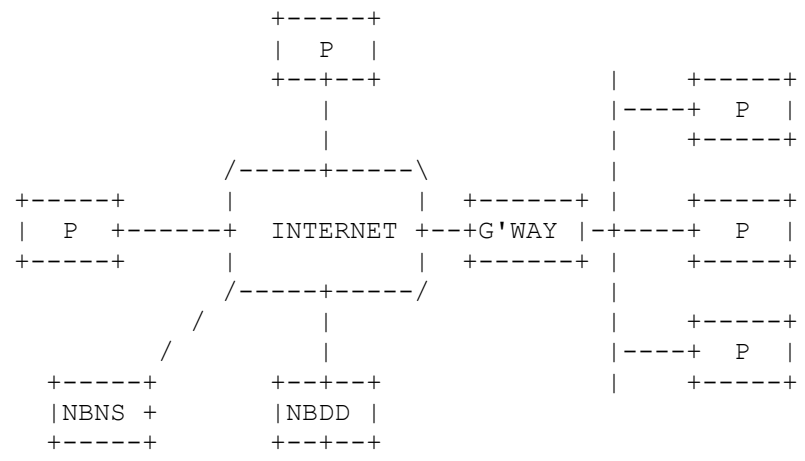
P Nodes Only
Mixed M and P Nodes

RFC-1001 Protocol Standard for a NetBIOS Service - Internet Topologies

P Nodes Only

P nodes may be scattered at various locations in an internetwork. They require both an NBNS and an NBDD for NetBIOS name and datagram support, respectively.

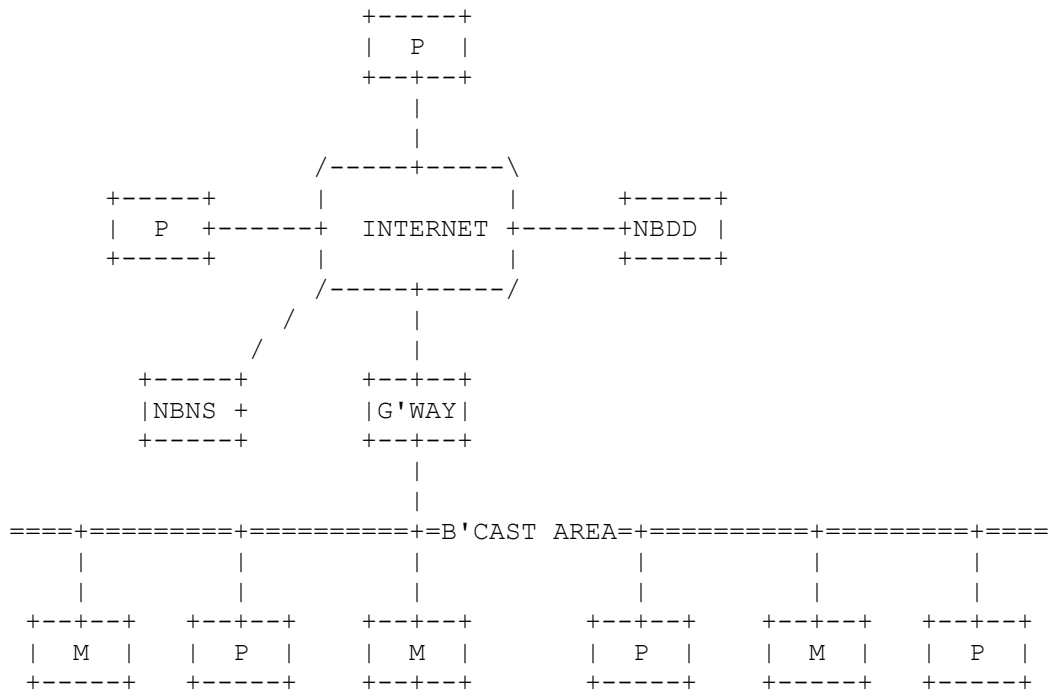
The NetBIOS scope is determined by the NetBIOS scope identifier (domain name) used by the various P (and M) nodes. An internet may contain numerous NetBIOS scopes.



Any P node may be replaced by an M node with no loss of function to any node. However, broadcast activity will be increased in the broadcast area to which the M node is attached.

Mixed M and P Nodes

M and P nodes may be mixed. When locating NetBIOS names, M nodes will tend to find names held by other M nodes on the same common broadcast area in preference to names held by P nodes or M nodes elsewhere in the network.



NOTE: B and M nodes should not be intermixed in an internet environment. Doing so would allow undetected NetBIOS name conflicts to arise and cause unpredictable behavior.

General Methods

Overlying the specific protocols, described later, are a few general methods of interaction between entities.

Request/Response Interaction Style

Most interactions between entities consist of a request flowing in one direction and a subsequent response flowing in the opposite direction.

In those situations where interactions occur on unreliable transports (i.e. UDP) or when a request is broadcast, there may not be a strict interlocking or one-to-one relationship between requests and responses. In no case, however, is more than one response generated for a received request. While a response is pending the responding entity may send one or more wait acknowledgements.

Retransmission of Requests

UDP is an unreliable delivery mechanism where packets can be lost, received out of transmit sequence, duplicated and delivery can be significantly delayed. Since the NetBIOS protocols make heavy use of UDP, they have compensated for its unreliability with extra mechanisms.

Each NetBIOS packet contains all the necessary information to process it. None of the protocols use multiple UDP packets to convey a single request or response. If more information is required than will fit in a single UDP packet, for example, when a P-type node wants all the owners of a group name from a NetBIOS server, a TCP connection is used. Consequently, the NetBIOS protocols will not fail because of out of sequence delivery of UDP packets.

To overcome the loss of a request or response packet, each request operation will retransmit the request if a response is not received within a specified time limit.

Protocol operations sensitive to successive response packets, such as name conflict detection, are protected from duplicated packets because they ignore successive packets with the same NetBIOS information. Since no state on the responder's node is associated with a request, the responder just sends the appropriate response whenever a request packet arrives. Consequently, duplicate or delayed request packets have no impact.

For all requests, if a response packet is delayed too long another request packet will be transmitted. A second response packet being sent in response to the second request packet is equivalent to a duplicate packet. Therefore, the protocols will ignore the second packet received. If the delivery of a response is delayed until after the request operation has been completed, successfully or not, the response packet is ignored.

Requests Without Responses: Demands

Some request types do not have matching responses. These requests are known as "demands". In general a "demand" is an imperative request; the receiving node is expected to obey. However, because demands are unconfirmed, they are used only in situations where, at most, limited damage would occur if the demand packet should be lost.

Demand packets are not retransmitted.

Transactions

Interactions between a pair of entities are grouped into "transactions". These transactions comprise one or more request/response pairs.

Transaction ID

Since multiple simultaneous transactions may be in progress between a pair of entities a "transaction id" is used.

The originator of a transaction selects an ID unique to the originator. The transaction id is reflected back and forth in each interaction within the transaction. The transaction partners must match responses and requests by comparison of the transaction ID and the IP address of the transaction partner. If no matching request can be found the response must be discarded.

A new transaction ID should be used for each transaction. A simple 16 bit transaction counter ought to be an adequate id generator. It is probably not necessary to search the space of outstanding transaction ID to filter duplicates: it is extremely unlikely that any transaction will have a lifetime that is more than a small fraction of the typical counter cycle period. Use of the IP addresses in conjunction with the transaction ID further reduces the possibility of damage should transaction IDs be prematurely re-used.

TCP and UDP Foundations

This version of the NetBIOS-over-TCP protocols uses UDP for many interactions. In the future this RFC may be extended to permit such interactions to occur over TCP connections (perhaps to increase efficiency when multiple interactions occur within a short time or when NetBIOS datagram traffic reveals that an application is using NetBIOS datagrams to support connection- oriented service.)

Representation of NetBIOS Names

NetBIOS names as seen across the client interface to NetBIOS are exactly 16 bytes long. Within the NetBIOS-over-TCP protocols, a longer representation is used.

There are two levels of encoding. The first level maps a NetBIOS name into a domain system name. The second level maps the domain system name into the "compressed" representation required for interaction with the domain name system.

Except in one packet, the second level representation is the only NetBIOS name representation used in NetBIOS-over-TCP packet formats. The exception is the RDATA field of a NODE STATUS RESPONSE packet.

RFC-1001 Protocol Standard for a NetBIOS Service - Representation of Names

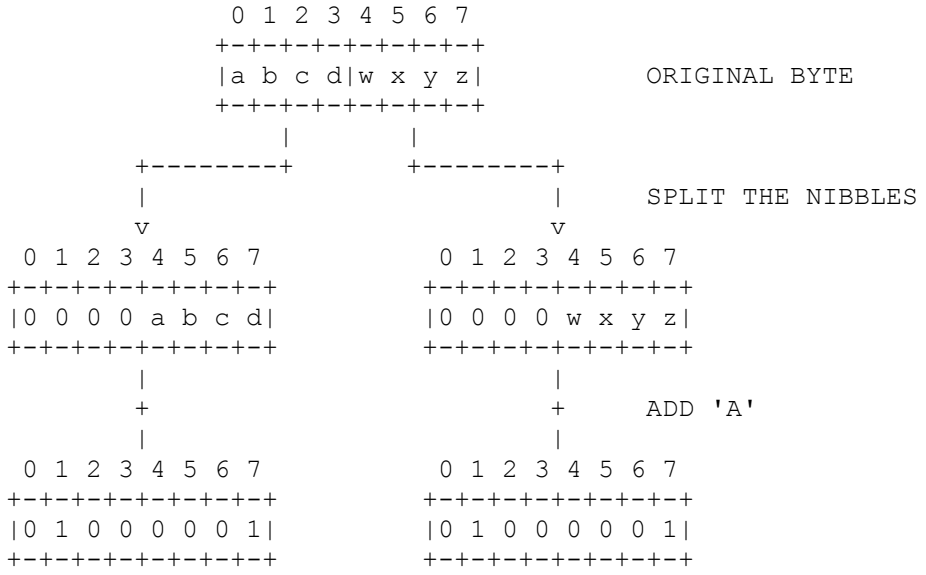
First Level Encoding

The first level representation consists of two parts:

- NetBIOS name
- NetBIOS scope identifier

The 16 byte NetBIOS name is mapped into a 32 byte wide field using a reversible, half-ASCII, biased encoding. Each half-octet of the NetBIOS name is encoded into one byte of the 32 byte field. The first half octet is encoded into the first byte, the second half- octet into the second byte, etc.

Each 4-bit, half-octet of the NetBIOS name is treated as an 8-bit, right-adjusted, zero-filled binary number. This number is added to value of the ASCII character 'A' (hexidecimal 41). The resulting 8- bit number is stored in the appropriate byte. The following diagram demonstrates this procedure:



This encoding results in a NetBIOS name being represented as a sequence of 32 ASCII, upper-case characters from the set {A,B,C...N,O,P}.

The NetBIOS scope identifier is a valid domain name (without a leading dot).

An ASCII dot (2E hexidecimal) and the scope identifier are appended to the encoded form of the NetBIOS name, the result forming a valid domain name.

For example, the NetBIOS name "The NetBIOS name" in the NetBIOS scope "SCOPE.ID.COM" would be represented at level one by the ASCII character string:

FEGHGFCAEOGFHEECEJEPFDCAHEGBGNGF.SCOPE.ID.COM

Second Level Encoding

The first level encoding must be reduced to second level encoding. This is performed according to the rules defined in on page 31 of RFC 883[12] in the section on "Domain name representation and compression". Also see the section titled "Name Formats" in the Detailed Specifications[RFC-1002].

RFC-1001 Protocol Standard for a NetBIOS Service - Concepts and Methods

NetBIOS Name Service

Before a name may be used, the name must be registered by a node. Once acquired, the name must be defended against inconsistent registration by other nodes. Before building a NetBIOS session or sending a NetBIOS datagram, the one or more holders of the name must be located.

The NetBIOS name service is the collection of procedures through which nodes acquire, defend, and locate the holders of NetBIOS names.

The name service procedures are different depending whether the end- node is of type B, P, or M.

RFC-1001 Protocol Standard for a NetBIOS Service - Name Service

Overview of NetBIOS Name Service

RFC-1001 Protocol Standard for a NetBIOS Service - Name Service

Name Registration (CLAIM)

Each NetBIOS node can own more than one name. Names are acquired dynamically through the registration (name claim) procedures.

Every node has a permanent unique name. This name, like any other name, must be explicitly registered by all end-node types.

A name can be unique (exclusive) or group (non-exclusive). A unique name may be owned by a single node; a group name may be owned by any number of nodes. A name ceases to exist when it is not owned by at least one node. There is no intrinsic quality of a name which determines its characteristics: these are established at the time of registration.

Each node maintains state information for each name it has registered. This information includes:

- Whether the name is a group or unique name
- Whether the name is "in conflict"
- Whether the name is in the process of being deleted

See Packet Format.

B nodes perform name registration by broadcasting claim requests, soliciting a defense from any node already holding the name.

P nodes perform name registration through the agency of the NBNS.

M nodes register names through an initial broadcast, like B nodes, then, in the absence of an objection, by following the same procedures as a P node. In other words, the broadcast action may terminate the attempt, but is not sufficient to confirm the registration.

RFC-1001 Protocol Standard for a NetBIOS Service - Name Service

Name Query (Discovery)

Name query (also known as "resolution" or "discovery") is the procedure by which the IP address(es) associated with a NetBIOS name are discovered. Name query is required during the following operations:

During session establishment, calling and called names must be specified. The calling name must exist on the node that posts the CALL. The called name must exist on a node that has previously posted a LISTEN. Either name may be a unique or group name.

When a directed datagram is sent, a source and destination name must be specified. If the destination name is a group name, a datagram is sent to all the members of that group.

Different end-node types perform name resolution using different techniques, but using the same packet formats:

- B nodes solicit name information by broadcasting a request.
- P nodes ask the NBNS.
- M nodes broadcast a request. If that does not provide the desired information, an inquiry is sent to the NBNS.

Name Release

NetBIOS names may be released explicitly or silently by an end- node. Silent release typically occurs when an end-node fails or is turned- off. Most of the mechanisms described below are present to detect silent name release. A common packet format is used.

Explicit Release

B nodes explicitly release a name by broadcasting a notice.

P nodes send a notification to their NBNS.

M nodes both broadcast a notice and inform their supporting NBNS.

Name Lifetime and Refresh

Names held by an NBNS are given a lifetime during name registration. The NBNS will consider a name to have been silently released if the end-node fails to send a name refresh message to the NBNS before the lifetime expires. A refresh restarts the lifetime clock.

NOTE: The implementor should be aware of the tradeoff between accuracy of the database and the internet overhead that the refresh mechanism introduces. The lifetime period should be tuned accordingly.

For group names, each end-node must send refresh messages. A node that fails to do so will be considered to have silently released the name and dropped from the group.

The lifetime period is established through a simple negotiation mechanism during name registration: In the name registration request, the end-node proposes a lifetime value or requests an infinite lifetime. The NBNS places an actual lifetime value into the name registration response. The NBNS is always allowed to respond with an infinite actual period. If the end node proposed an infinite lifetime, the NBNS may respond with any definite period. If the end node proposed a definite period, the NBNS may respond with any definite period greater than or equal to that proposed.

This negotiation of refresh times gives the NBNS means to disable or enable refresh activity. The end-nodes may set a minimum refresh cycle period.

NBNS implementations which are completely reliable may disable refresh.

Name Challenge Overview

To detect whether a node has silently released its claim to a name, it is necessary on occasion to challenge that node's current ownership. If the node defends the name then the node is allowed to continue possession. Otherwise it is assumed that the node has released the name. See packet format.

A name challenge may be issued by an NBNS or by a P or M node. A challenge may be directed towards any end-node type: B, P, or M.

Group Name Fade-Out

NetBIOS groups may contain an arbitrarily large number of members. The time to challenge all members could be quite large.

To avoid long delays when names are claimed through an NBNS, an optimistic heuristic has been adopted. It is assumed that there will always be some node which will defend a group name. Consequently, it is recommended that the NBNS will immediately reject a claim request for a unique name when there already exists a group with the same name. The NBNS will never return an IP address (in response to a NAME REGISTRATION REQUEST) when a group name exists.

An NBNS will consider a group to have faded out of existence when the last remaining member fails to send a timely refresh message or explicitly releases the name.

RFC-1001 Protocol Standard for a NetBIOS Service - Name Service

Name Conflict

Name conflict exists when a unique name has been claimed by more than one node on a NetBIOS network. B, M, and NBNS nodes may detect a name conflict. The detection mechanism used by B and M nodes is active only during name discovery. The NBNS may detect conflict at any time it verifies the consistency of its name database.

B and M nodes detect conflict by examining the responses received in answer to a broadcast name query request. The first response is taken as authoritative. Any subsequent, inconsistent responses represent conflicts. See packet format.

Subsequent responses are inconsistent with the authoritative response when:

The subsequent response has the same transaction ID as the NAME QUERY REQUEST.

and

The subsequent response is not a duplicate of the authoritative response.

and either:

The group/unique characteristic of the authoritative response is "unique".

or

The group/unique characteristic of the subsequent response is "unique".

The period in which B and M nodes examine responses is limited by a conflict timer, CONFLICT_TIMER. The accuracy or duration of this timer is not crucial: the NetBIOS system will continue to operate even with persistent name conflicts.

Conflict conditions are signaled by sending a NAME CONFLICT DEMAND to the node owning the offending name. Nothing is sent to the node which originated the authoritative response.

Any end-node that receives NAME CONFLICT DEMAND is required to update its "local name table" to reflect that the name is in conflict. (The "local name table" on each node contains names that have been successfully registered by that node.)

Notice that only those nodes that receive the name conflict message place a conflict mark next to a name.

Logically, a marked name does not exist on that node. This means that the node should not defend the name (for name claim purposes), should not respond to a name discovery requests for that name, nor should the node send name refresh messages for that name. Furthermore, it can no longer be used by that node for any session establishment or sending or receiving datagrams. Existing sessions are not affected at the time a name is marked as being in conflict.

The only valid user function against a marked name is DELETE NAME. Any other user NetBIOS function returns immediately with an error code of "NAME CONFLICT".

Adapter Status

An end-node or the NBNS may ask any other end-node for a collection of information about the NetBIOS status of that node. This status consists of, among other things, a list of the names which the node believes it owns. The returned status is filtered to contain only those names which have the same NetBIOS scope identifier as the requestor's name.

When requesting node status, the requestor identifies the target node by NetBIOS name A name query transaction may be necessary to acquire the IP address for the name. Locally cached name information may be used in lieu of a query transaction. The requesting node sends a NODE STATUS REQUEST. In response, the receiving node sends a NODE STATUS RESPONSE containing its local name table and various statistics.

The amount of status which may be returned is limited by the size of a UDP packet. However, this is sufficient for the typical NODE STATUS RESPONSE packet.

End-Node NBNS Interaction

There are certain characteristics of end-node to NBNS interactions which are in common and are independent of any particular transaction type.

UDP, TCP, and Truncation

For all transactions between an end-node and an NBNS, either UDP or TCP may be used as a transport. If the NBNS receives a UDP based request, it will respond using UDP. If the amount of information exceeds what fits into a UDP packet, the response will contain a "truncation flag". In this situation, the end- node may open a TCP connection to the NBNS, repeat the request, and receive a complete, untruncated response.

NBNS WACK

While a name service request is in progress, the NBNS may issue a WAIT FOR ACKNOWLEDGEMENT RESPONSE (WACK) to assure the client end- node that the NBNS is still operational and is working on the request.

NBNS Redirection

The NBNS, because it follows Domain Name system styles of interaction, is permitted to redirect a client to another NBNS.

Secured Versus Non-Secured NBNS

An NBNS may be implemented in either of two general ways: The NBNS may monitor, and participate in, name activity to ensure consistency. This would be a "secured" style NBNS. Alternatively, an NBNS may be implemented to be essentially a "bulletin board" on which name information is posted and responsibility for consistency is delegated to the end-nodes. This would be a "non-secured" style NBNS.

Consistency of the NBNS Data Base

Even in a properly running NetBIOS scope the NBNS and its community of end-nodes may occasionally lose synchronization with respect to the true state of name registrations.

This may occur should the NBNS fail and lose all or part of its database.

More commonly, a P or M node may be turned-off (thus forgetting the names it has registered) and then be subsequently turned back on.

Finally, errors may occur or an implementation may be incorrect.

Various approaches have been incorporated into the NetBIOS-over- TCP protocols to minimize the impact of these problems.

1. The NBNS (or any other node) may "challenge" (using a NAME QUERY REQUEST) an end-node to verify that it actually owns a name.

   Such a challenge may occur at any time. Every end-node must be prepared to make a timely response.

   Failure to respond causes the NBNS to consider that the end-node has released the name in question.

   (If UDP is being used as the underlying transport, the challenge, like all other requests, must be retransmitted some number of times in the absence of a response.)

2. The NBNS (or any other node) may request (using the NODE STATUS REQUEST) that an end-node deliver its entire name table.

   This may occur at any time. Every end-node must be prepared to make a timely response.

   Failure to respond permits (but does not require) the NBNS to consider that the end-node has failed and released all names to which it had claims. (Like the challenge, on a UDP transport, the request must be retransmitted in the absence of a response.)

3. The NBNS may revoke a P or M node's use of a name by sending either a NAME CONFLICT DEMAND or a NAME RELEASE REQUEST to the node.

   The receiving end-node may continue existing sessions which use that name, but must otherwise cease using that name. If the NBNS placed the name in conflict, the name may be re- acquired only by deletion and subsequent reclamation. If the NBNS requested that the name be released, the node may attempt to re-acquire the name without first performing a name release transaction.

4. The NBNS may impose a "time-to-live" on each name it registers. The registering node is made aware of this time value during the name registration procedure.

   Simple or reliable NBNS's may impose an infinite time-to- live.

5. If an end-node holds any names that have finite time-to- live values, then that node must periodically send a status report to the NBNS. Each name is reported using the NAME REFRESH REQUEST packet.

   These status reports restart the timers of both the NBNS and the reporting node. However, the only timers which are restarted are

those associated with the name found in the status report. Timers on other names are not affected.

The NBNS may consider that a node has released any name which has not been refreshed within some multiple of name's time-to-live.

A well-behaved NBNS, would, however, issue a challenge to-, or request a list of names from-, the non-reporting end- node before deleting its name(s). The absence of a response, or of the name in a response, will confirm the NBNS decision to delete a name.

6. The absence of reports may cause the NBNS to infer that the end-node has failed. Similarly, receipt of information widely divergent from what the NBNS believes about the node, may cause the NBNS to consider that the end-node has been restarted.

   The NBNS may analyze the situation through challenges or requests for a list of names.

7. A very cautious NBNS is free to poll nodes (by sending NAME QUERY REQUEST or NODE STATUS REQUEST packets) to verify that their name status is the same as that registered in the NBNS.

   NOTE: Such polling activity, if used at all by an implementation, should be kept at a very low level or enabled only during periods when the NBNS has some reason to suspect that its information base is inaccurate.

8. P and M nodes can detect incorrect name information at session establishment.

   If incorrect information is found, NBNS is informed via a NAME RELEASE REQUEST originated by the end-node which detects the error.

Name Caching

An end-node may keep a local cache of NetBIOS name-to-IP address translation entries.

All cache entries should be flushed on a periodic basis.

In addition, a node ought to flush any cache information associated with an IP address if the node receives any information indicating that there may be any possibility of trouble with the node at that IP address. For example, if a NAME CONFLICT DEMAND is sent to a node, all cached information about that node should be cleared within the sending node.

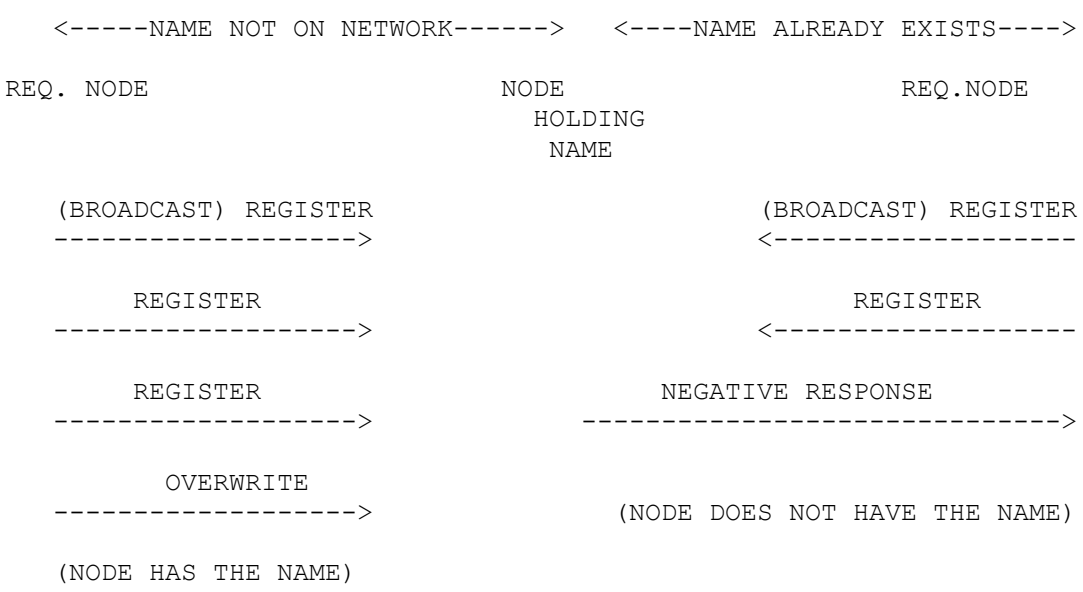
Name Registration Transactions

Name Registration by B Nodes

A name claim transaction initiated by a B node is broadcast throughout the broadcast area. The NAME REGISTRATION REQUEST will be heard by all B and M nodes in the area. Each node examines the claim to see whether it it is consistent with the names it owns. If an inconsistency exists, a NEGATIVE NAME REGISTRATION RESPONSE is unicast to the requestor. The requesting node obtains ownership of the name (or membership in the group) if, and only if, no NEGATIVE NAME REGISTRATION RESPONSEs are received within the name claim timeout, CONFLICT_TIMER. (See "Defined Constants and Variables" in the Detailed Specification for the value of this timer.)

A B node proclaims its new ownership by broadcasting a NAME OVERWRITE DEMAND.

B-Node Registration Process



The NAME REGISTRATION REQUEST, like any request, must be repeated if no response is received within BCAST_REQ_RETRY_TIMEOUT. Transmission of the request is attempted BCAST_REQ_RETRY_COUNT times.

Name Registration By P Nodes

A name registration may proceed in various ways depending whether the name being registered is new to the NBNS. If the name is known to the NBNS, then challenges may be sent to the prior holder(s).

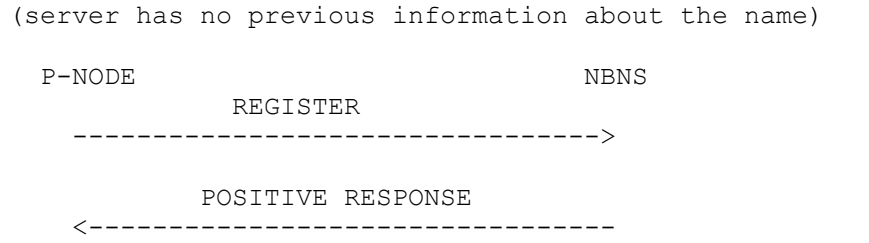
RFC-1001 Protocol Standard for a NetBIOS Service - Name Service

New Name, or New Group Member

The diagram, below, shows the sequence of events when an end-node registers a name which is new to the NBNS. (The diagram omits WACKs, NBNS redirections, and retransmission of requests.)

This same interaction will occur if the name being registered is a group name and the group already exists. The NBNS will add the registrant to the set of group members.

P-Node Registration Process



The interaction is rather simple: the end-node sends a NAME REGISTRATION REQUEST, the NBNS responds with a POSITIVE NAME REGISTRATION RESPONSE.

RFC-1001 Protocol Standard for a NetBIOS Service - Name Service

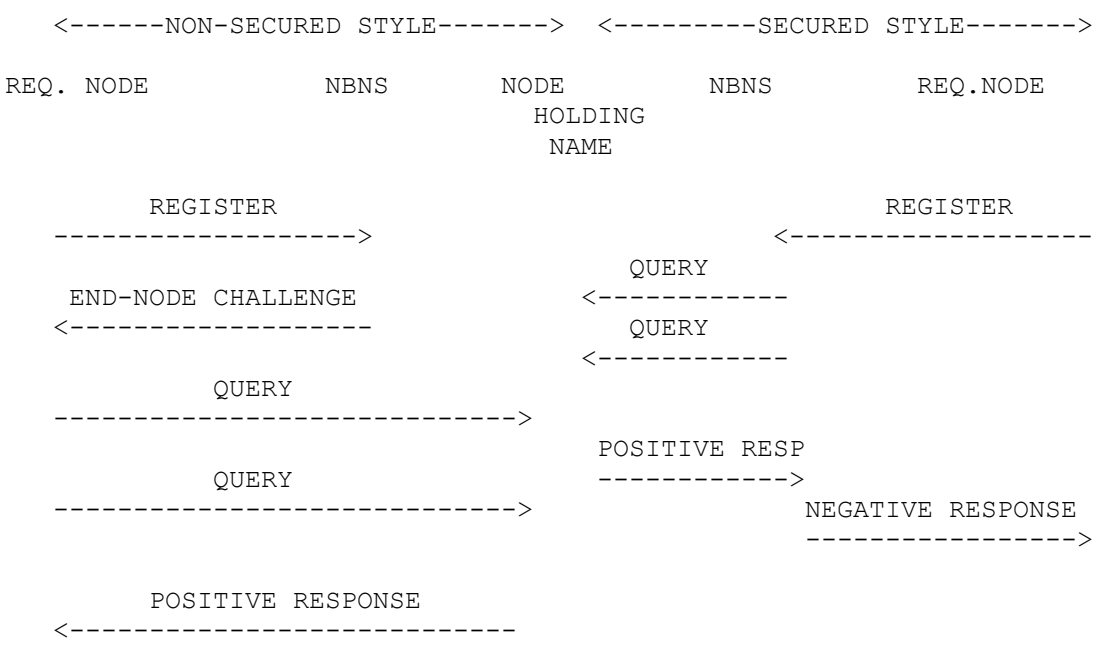
Existing Name and Owner is Still Active

The following diagram shows interactions when an attempt is made to register a unique name, the NBNS is aware of an existing owner, and that existing owner is still active.

There are two sides to the diagram. The left side shows how a non- secured NBNS would handle the matter. Secured NBNS activity is shown on the right.

P-Node Registration Process

(server HAS a previous owner that IS active)



A non-secured NBNS will answer the NAME REGISTRATION REQUEST with a END-NODE CHALLENGE REGISTRATION RESPONSE. This response asks the end-node to issue a challenge transaction against the node indicated in the response. In this case, the prior node will defend against the challenge and the registering end-node will simply drop the registration attempt without further interaction with the NBNS.

A secured NBNS will refrain from answering the NAME REGISTRATION REQUEST until the NBNS has itself challenged the prior holder(s) of the name. In this case, the NBNS finds that that the name is still being defended and consequently returns a NEGATIVE NAME REGISTRATION RESPONSE to the registrant.

Due to the potential time for the secured NBNS to make the challenge(s), it is likely that a WACK will be sent by the NBNS to the registrant.

Although not shown in the diagram, a non-secured NBNS will send a NEGATIVE NAME REGISTRATION RESPONSE to a request to register a unique name when there already exists a group of the same name. A secured NBNS may elect to poll (or challenge) the group members to determine whether any active members remain. This may impose a heavy load on the network. It is recommended that group names be allowed to fade- out through the name refresh mechanism.

RFC-1001 Protocol Standard for a NetBIOS Service - Name Service

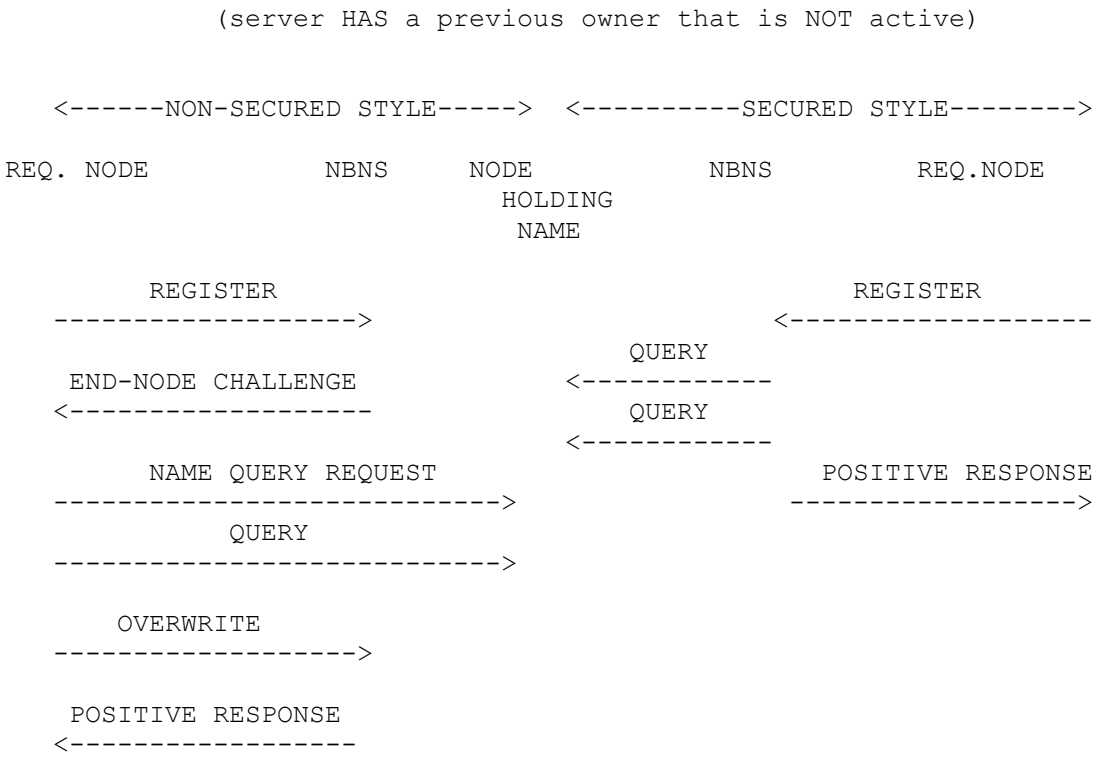
Existing Name and Owner is Inactive

The following diagram shows interactions when an attempt is made to register a unique name, the NBNS is aware of an existing owner, and that existing owner is no longer active.

A non-secured NBNS will answer the NAME REGISTRATION REQUEST with a END-NODE CHALLENGE REGISTRATION RESPONSE. This response asks the end-node to issue a challenge transaction against the node indicated in the response. In this case, the prior node will not defend against the challenge. The registrant will inform the NBNS through a NAME OVERWRITE REQUEST. The NBNS will replace the prior name information in its database with the information in the overwrite request.

A secured NBNS will refrain from answering the NAME REGISTRATION REQUEST until the NBNS has itself challenged the prior holder(s) of the name. In this case, the NBNS finds that that the name is not being defended and consequently returns a POSITIVE NAME REGISTRATION RESPONSE to the registrant.

P-Node Registration Process



Due to the potential time for the secured NBNS to make the challenge(s), it is likely that a WACK will be sent by the NBNS to the registrant.

A secured NBNS will immediately send a NEGATIVE NAME REGISTRATION RESPONSE in answer to any NAME OVERWRITE REQUESTS it may receive.

Name Registration by M Nodes

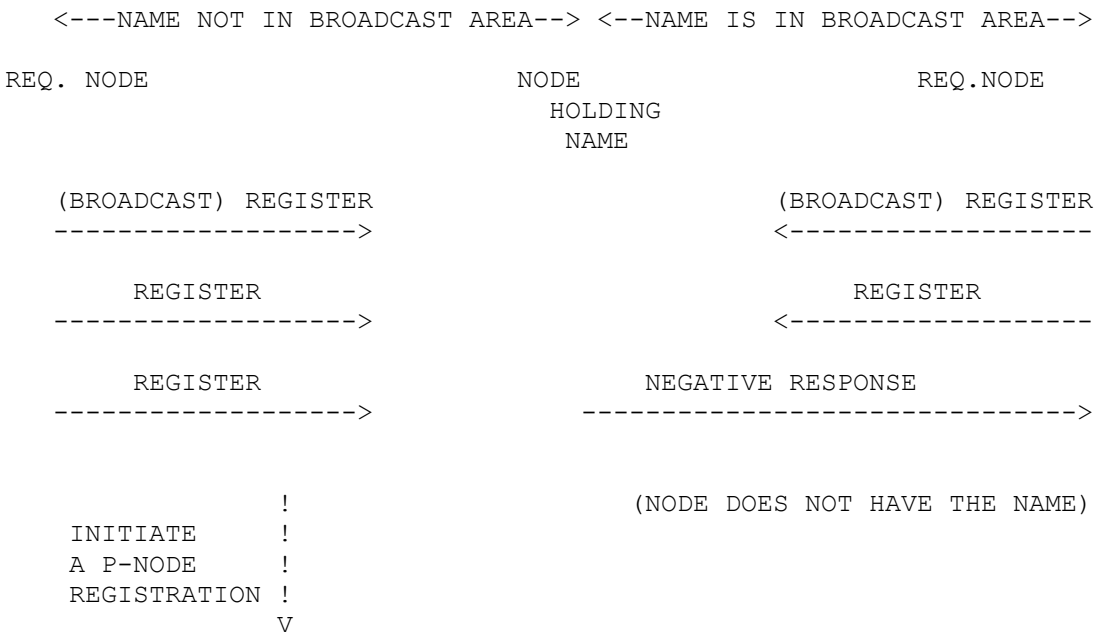
An M node begin a name claim operation as if the node were a B node: it broadcasts a NAME REGISTRATION REQUEST and listens for NEGATIVE NAME REGISTRATION RESPONSEs. Any NEGATIVE NAME REGISTRATION RESPONSE prevents the M node from obtaining the name and terminates the claim operation.

If, however, the M node does not receive any NEGATIVE NAME REGISTRATION RESPONSE, the M node must continue the claim procedure as if the M node were a P node.

Only if both name claims were successful does the M node acquire the name.

The following diagram illustrates M node name registration:

M-Node Registration Process



Name Query Transactions

Name query transactions are initiated by end-nodes to obtain the IP address(es) and other attributes associated with a NetBIOS name.

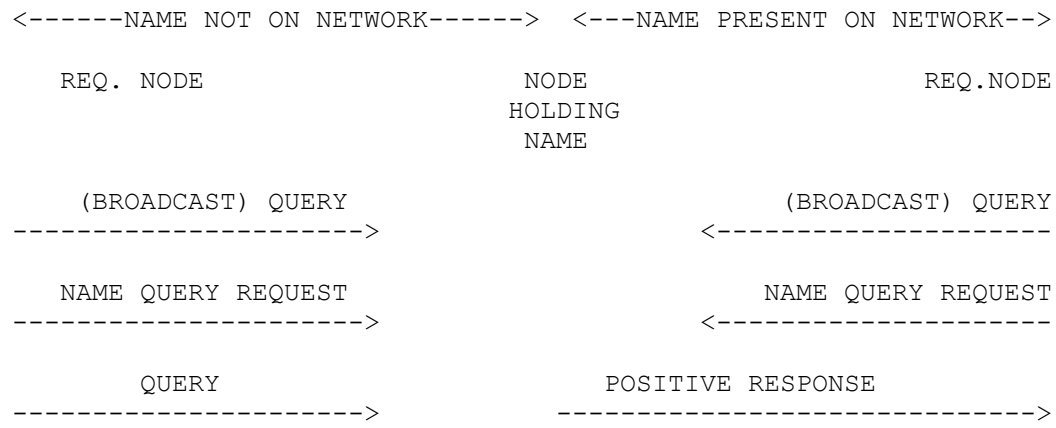
Query by B Nodes

The following diagram shows how B nodes go about discovering who owns a name.

The left half of the diagram illustrates what happens if there are no holders of the name. In that case no responses are received in answer to the broadcast NAME QUERY REQUEST(s).

The right half shows a POSITIVE NAME QUERY RESPONSE unicast by a name holder in answer to the broadcast request. A name holder will make this response to every NAME QUERY REQUEST that it hears. And each holder acts this way. Thus, the node sending the request may receive many responses, some duplicates, and from many nodes.

B-Node Discovery Process



Name query is generally, but not necessarily, a prelude to NetBIOS session establishment or NetBIOS datagram transmission. However, name query may be used for other purposes.

A B node may elect to build a group membership list for subsequent use (e.g. for session establishment) by collecting and saving the responses.

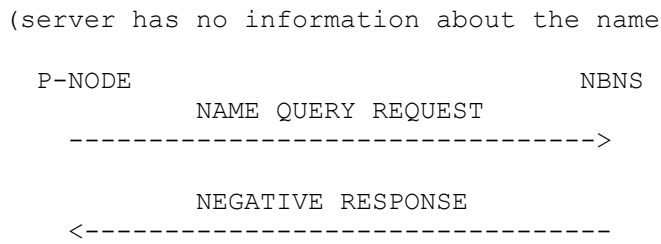
Query by P Nodes

An NBNS answers queries from a P node with a list of IP address and other information for each owner of the name. If there are multiple owners (i.e. if the name is a group name), the NBNS loads as many answers into the response as will fit into a UDP packet. A truncation flag indicates whether any additional owner information remains. All the information may be obtained by repeating the query over a TCP connection.

The NBNS is not required to impose any order on its answer list.

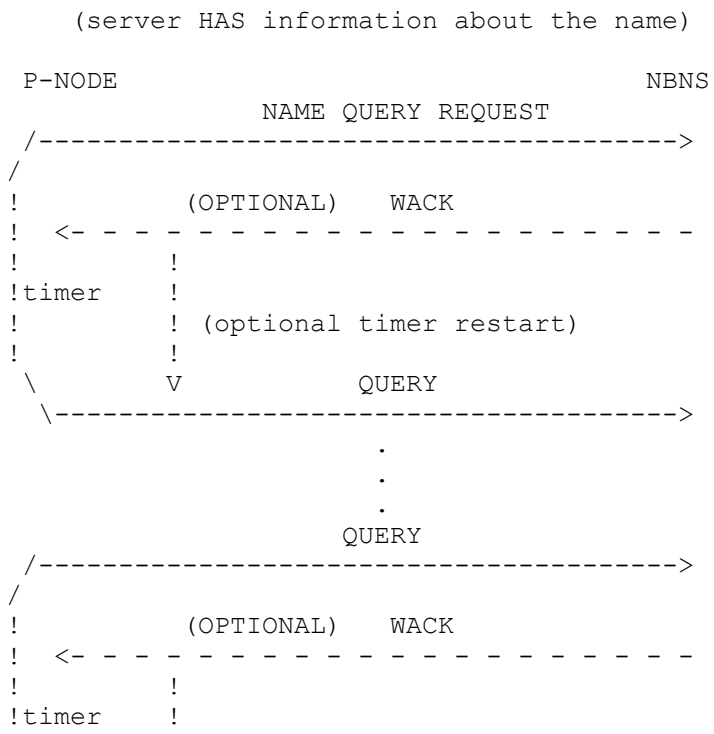
The following diagram shows what happens if the NBNS has no information about the name:

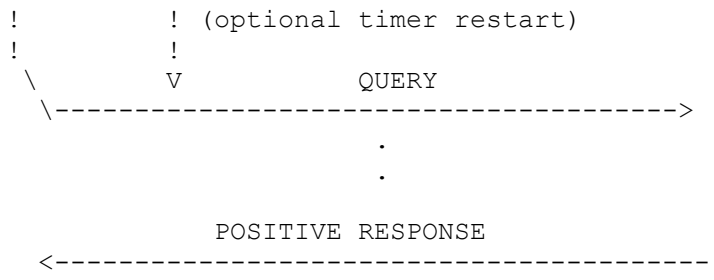
P-Node Discovery Process



The next diagram illustrates interaction between the end-node and the NBNS when the NBNS does have information about the name. This diagram shows, in addition, the retransmission of the request by the end-node in the absence of a timely response. Also shown are WACKs (or temporary, intermediate responses) sent by the NBNS to the end-node:

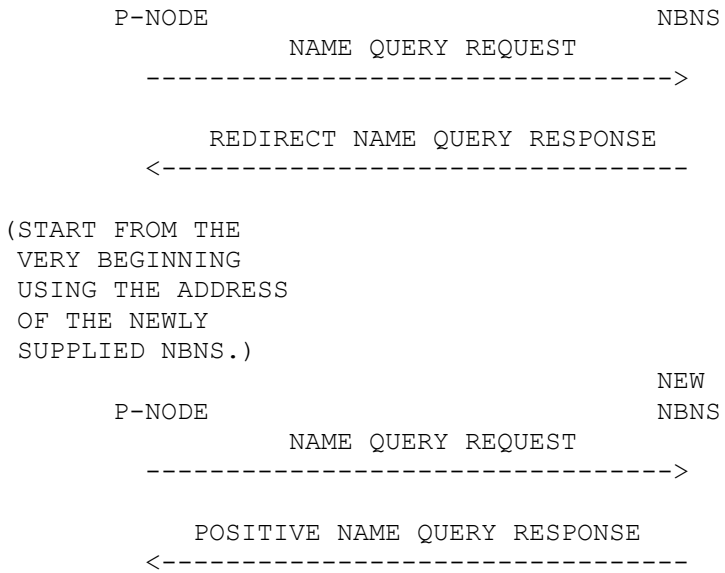
P-Node Query Process





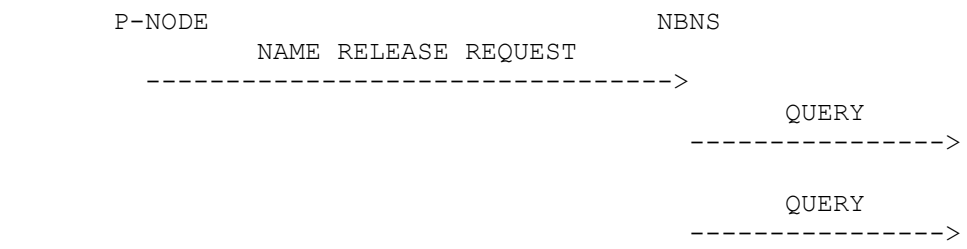
The following diagram illustrates NBNS redirection. Upon receipt of a NAME QUERY REQUEST, the NBNS redirects the client to another NBNS. The client repeats the request to the new NBNS and obtains a response. The diagram shows that response as a POSITIVE NAME QUERY RESPONSE. However any legal NBNS response may occur in actual operation.

NBNS Redirection



The next diagram shows how a P or M node tells the NBNS that the NBNS has provided incorrect information. This procedure may begin after a DATAGRAM ERROR packet has been received or a session set-up attempt has discovered that the NetBIOS name does not exist at the destination, the IP address of which was obtained from the NBNS during a prior name query transaction. The NBNS, in this case a secure NBNS, issues queries to verify whether the information is, in fact, incorrect. The NBNS closes the transaction by sending either a POSITIVE or NEGATIVE NAME RELEASE RESPONSE, depending on the results of the verification.

Correcting NBNS Information Base



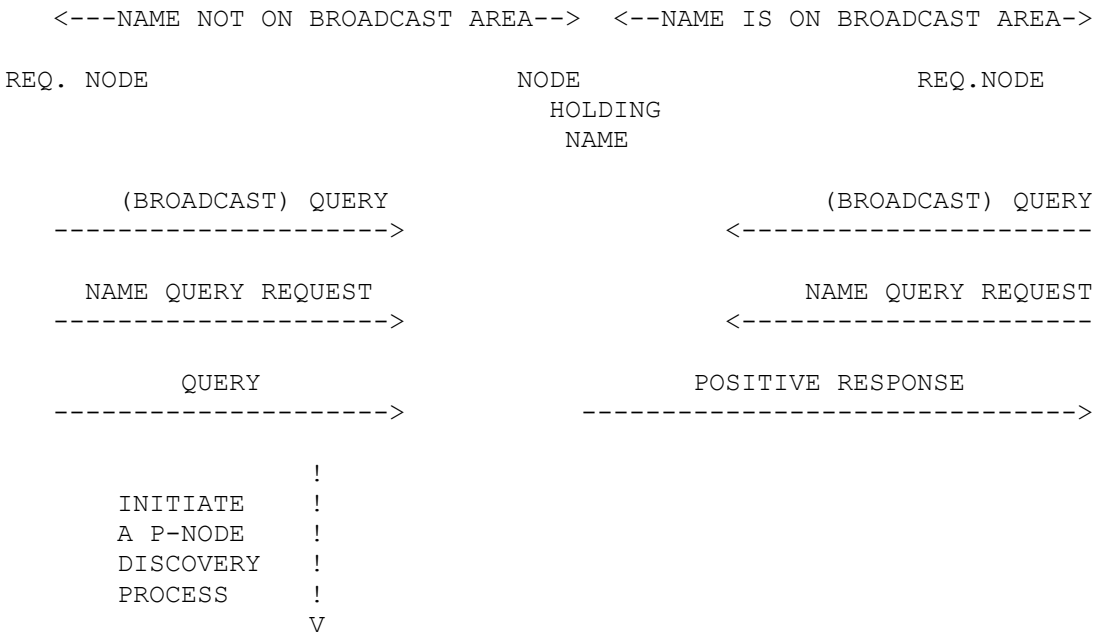
(NAME TAKEN OFF THE DATABASE
IF NBNS FINDS IT TO BE
INCORRECT)

POSITIVE/NEGATIVE RESPONSE
<----------

Query by M Nodes

M node name query follows the B node pattern. In the absence of adequate results, the M node then continues by performing a P node type query. This is shown in the following diagram:

M-Node Discovery Process



Acquire Group Membership List

The entire membership of a group may be acquired by sending a NAME QUERY REQUEST to the NBNS. The NBNS will respond with a POSITIVE NAME QUERY RESPONSE or a NEGATIVE NAME QUERY RESPONSE. A negative response completes the procedure and indicates that there are no members in the group.

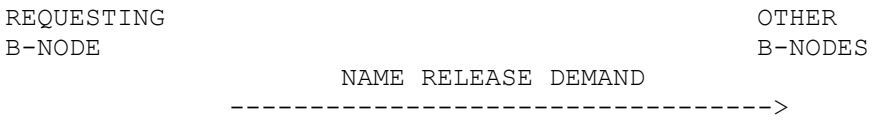
If the positive response has the truncation bit clear, then the response contains the entire list of group members. If the truncation bit is set, then this entire procedure must be repeated, but using TCP as a foundation rather than UDP.

Name Release Transactions

Release by B Nodes

A NAME RELEASE DEMAND contains the following information:

- NetBIOS name
- The scope of the NetBIOS name
  - Name type: unique or group
  - IP address of the releasing node
  - Transaction ID



RFC-1001 Protocol Standard for a NetBIOS Service - Name Service

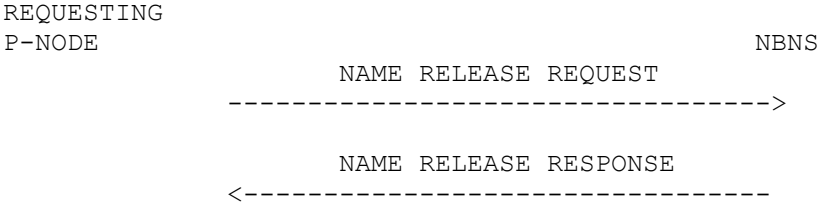
Release by P Nodes

A NAME RELEASE REQUEST contains the following information:

- NetBIOS name
- The scope of the NetBIOS name
  - Name type: unique or group
  - IP address of the releasing node
  - Transaction ID

A NAME RELEASE RESPONSE contains the following information:

- NetBIOS name
- The scope of the NetBIOS name
- Name type: unique or group
- IP address of the releasing node
- Transaction ID
- Result:
  - Yes: name was released
  - No: name was not released, a reason code is provided

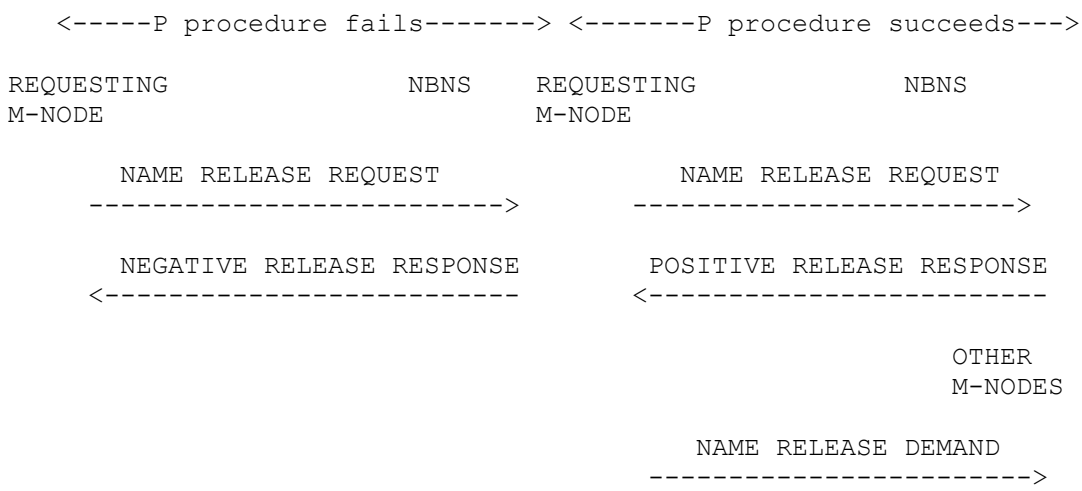


Release by M Nodes

The name release procedure of the M node is a combination of the P and B node name release procedures. The M node first performs the P release procedure. If the P procedure fails then the release procedure does not continue, it fails. If and only if the P procedure succeeds then the M node broadcasts the NAME RELEASE DEMAND to the broadcast area, the B procedure.

NOTE: An M node typically performs a B-style operation and then a P-style operation. In this case, however, the P-style operation comes first.

The following diagram illustrates the M node name release procedure:



Name Maintenance Transactions

Name Refresh

Name refresh transactions are used to handle the following situations:

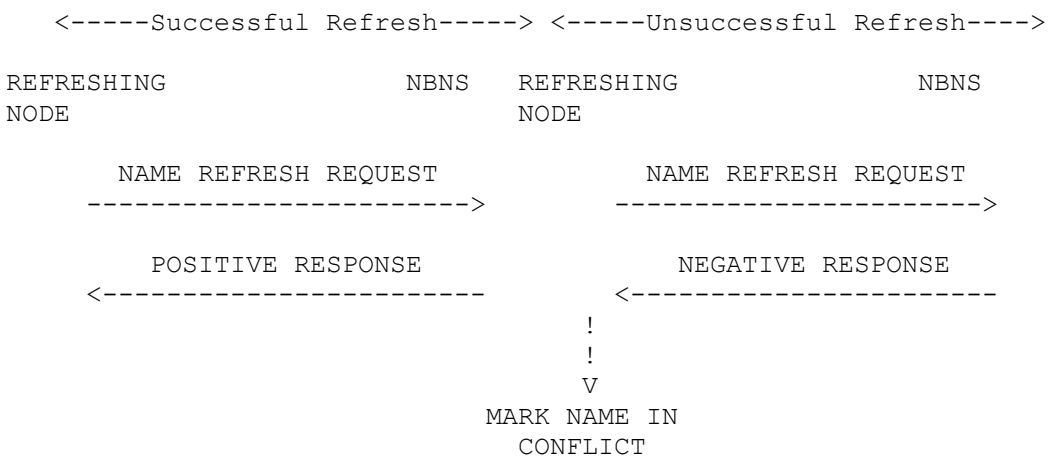
a) An NBNS node needs to detect if a P or M node has "silently" gone down, so that names held by that node can be purged from the data base.

b) If the NBNS goes down, it needs to be able to reconstruct the data base when it comes back up.

c) If the network should be partitioned, the NBNS needs to be able to able to update its data base when the network reconnects.

Each P or M node is responsible for sending periodic NAME REFRESH REQUESTs for each name that it has registered. Each refresh packet contains a single name that has been successfully registered by that node. The interval between such packets is negotiated between the end node and the NBNS server at the time that the name is initially claimed. At name claim time, an end node will suggest a refresh timeout value. The NBNS node can modify this value in the reply packet. A NBNS node can also choose to tell the end node to not send any refresh packet by using the "infinite" timeout value in the response packet. The timeout value returned by the NBNS is the actual refresh timeout that the end node must use.

When a node sends a NAME REFRESH REQUEST, it must be prepared to receive a negative response. This would happen, for example, if the the NBNS discovers that the the name had already been assigned to some other node. If such a response is received, the end node should mark the name as being in conflict. Such an entry should be treated in the same way as if name conflict had been detected against the name. The following diagram illustrates name refresh:



Name Challenge

Name challenge is done by sending a NAME QUERY REQUEST to an end node of any type. If a POSITIVE NAME QUERY RESPONSE is returned, then that node still owns the name. If a NEGATIVE NAME QUERY RESPONSE is received or if no response is received, it can be assumed that the end node no longer owns the name.

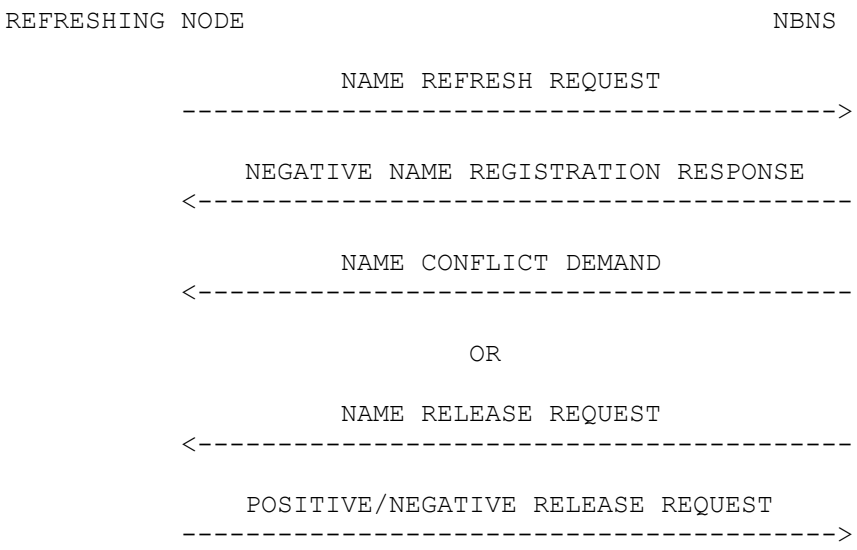
Name challenge can be performed either by the NBNS node, or by an end node. When an end-node sends a name claim packet, the NBNS node may do the challenge operation. The NBNS node can choose, however, to require the end node do the challenge. In that case, the NBNS will send an END-NODE CHALLENGE RESPONSE packet to the end node, which should then proceed to challenge the putative owner.

Note that the name challenge procedure sends a normal NAME QUERY REQUEST packet to the end node. It does not require a special packet. The only new packet introduced is the END-NODE CHALLENGE RESPONSE which is sent by an NBNS node when the NBNS wants the end- node to perform the challenge operation.

Clear Name Conflict

It is possible during a refresh request from a M or P node for a NBNS to detects a name in conflict. The response to the NAME REFRESH REQUEST must be a NEGATIVE NAME REGISTRATION RESPONSE. Optionally, in addition, the NBNS may send a NAME CONFLICT DEMAND or a NAME RELEASE REQUEST to the refreshing node. The NAME CONFLICT DEMAND forces the node to place the name in the conflict state. The node will eventually inform it's user of the conflict. The NAME RELEASE REQUEST will force the node to flush the name from its local name table completely. This forces the node to flush the name in conflict. This does not cause termination of existing sessions using this name.

The following diagram shows an NBNS detecting and correcting a conflict:



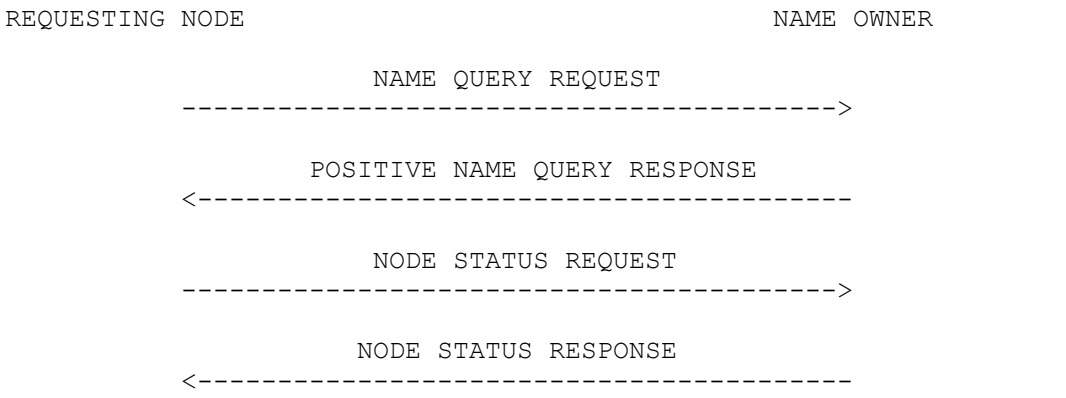
Adapter Status Transactions

Adapter status is obtained from a node as follows:

1. Perform a name discovery operation to obtain the IP addresses of a set of end-nodes.
2. Repeat until all end-nodes from the set have been used:
   a. Select one end-node from the set.
   b. Send a NODE STATUS REQUEST to that end-node using UDP.
   c. Await a NODE STATUS RESPONSE. (If a timely response is not forthcoming, repeat step "b" UCAST_REQ_RETRY_COUNT times. After the last retry, go to step "a".)
   d. If the truncation bit is not set in the response, the response contains the entire node status. Return the status to the user and terminate this procedure.
   e. If the truncation bit is set in the response, then not all status was returned because it would not fit into the response packet. The responder will set the truncation bit if the IP datagram length would exceed MAX_DATAGRAM_LENGTH. Return the status to the user and terminate this procedure.
3. Return error to user, no status obtained.

The repetition of step 2, above, through all nodes of the set, is optional.

Following is an example transaction of a successful Adapter Status operation:



NetBIOS Session Service

The NetBIOS session service begins after one or more IP addresses have been found for the target name. These addresses may have been acquired using the NetBIOS name query transactions or by other means, such as a local name table or cache.

NetBIOS session service transactions, packets, and protocols are identical for all end-node types. They involve only directed (point-to-point) communications.

RFC-1001 Protocol Standard for a NetBIOS Service - Session Service

Overview of NetBIOS Session Service

Session service has three phases:

Session establishment - it is during this phase that the IP address and TCP port of the called name is determined, and a TCP connection is established with the remote party.

Steady state - it is during this phase that NetBIOS data messages are exchanged over the session. Keep-alive packets may also be exchanged if the participating nodes are so configured.

Session close - a session is closed whenever either a party (in the session) closes the session or it is determined that one of the parties has gone down.

RFC-1001 Protocol Standard for a NetBIOS Service - Session Service

Session Establishment Phase Overview

An end-node begins establishment of a session to another node by somehow acquiring (perhaps using the name query transactions or a local cache) the IP address of the node or nodes purported to own the destination name.

Every end-node awaits incoming NetBIOS session requests by listening for TCP calls to a well-known service port, SSN_SRVC_TCP_PORT. Each incoming TCP connection represents the start of a separate NetBIOS session initiation attempt. The NetBIOS session server, not the ultimate application, accepts the incoming TCP connection(s).

Once the TCP connection is open, the calling node sends session service request packet. This packet contains the following information:

- Calling IP address (see note)
- Calling NetBIOS name
- Called IP address (see note)
- Called NetBIOS name

NOTE: The IP addresses are obtained from the TCP service interface.

When the session service request packet arrives at the NetBIOS server, one of the the following situations will exist:

- There exists a NetBIOS LISTEN compatible with the incoming call and there are adequate resources to permit session establishment to proceed.
- There exists a NetBIOS LISTEN compatible with the incoming call, but there are inadequate resources to permit establishment of a session.
- The called name does, in fact, exist on the called node, but there is no pending NetBIOS LISTEN compatible with the incoming call.
- The called name does not exist on the called node.

In all but the first case, a rejection response is sent back over the TCP connection to the caller. The TCP connection is then closed and the session phase terminates. Any retry is the responsibility of the caller. For retries in the case of a group name, the caller may use the next member of the group rather than immediately retrying the instant address. In the case of a unique name, the caller may attempt an immediate retry using the same target IP address unless the called name did not exist on the called node. In that one case, the NetBIOS name should be re-resolved.

If a compatible LISTEN exists, and there are adequate resources, then the session server may transform the existing TCP connection into the NetBIOS data session. Alternatively, the session server may redirect, or "retarget" the caller to another TCP port (and IP address).

If the caller is redirected, the caller begins the session establishment anew, but using the new IP address and TCP port given in the retarget response. Again a TCP connection is created, and again the calling and called node exchange credentials. The called party may accept the call, reject the call, or make a further redirection.

This mechanism is based on the presumption that, on hosts where it is not possible to transfer open TCP connections between processes, the host will have a central session server. Applications willing to receive NetBIOS calls will obtain an ephemeral TCP port number, post a TCP unspecified passive open on that port, and then pass that port number and NetBIOS name information to the NetBIOS session server using a NetBIOS LISTEN operation. When the call is placed, the session server will "retarget" the caller to the application's TCP socket. The caller will then place a new call, directly to the application.

The application has the responsibility to mimic the session server at least to the extent of receiving the calling credentials and then accepting or rejecting the call.

Retrying After Being Retargetted

A calling node may find that it can not establish a session with a node to which it was directed by the retargetting procedure. Since retargetting may be nested, there is an issue whether the caller should begin a retry at the initial starting point or back-up to an intermediate retargetting point. The caller may use any method. A discussion of two such methods is in Appendix B, "Retarget Algorithms".

Session Establishment to a Group Name

Session establishment with a group name requires special consideration. When a NetBIOS CALL attempt is made to a group name, name discovery will result in a list (possibly incomplete) of the members of that group. The calling node selects one member from the list and attempts to build a session. If that fails, the calling node may select another member and make another attempt.

When the session service attempts to make a connection with one of the members of the group, there is no guarantee that that member has a LISTEN pending against that group name, that the called node even owns, or even that the called node is operating.

Steady State Phase Overview

NetBIOS data messages are exchanged in the steady state. NetBIOS messages are sent by prepending the user data with a message header and sending the header and the user data over the TCP connection. The receiver removes the header and passes the data to the NetBIOS user.

In order to detect failure of one of the nodes or of the intervening network, "session keep alive" packets may be periodically sent in the steady state.

Any failure of the underlying TCP connection, whether a reset, a timeout, or other failure, implies failure of the NetBIOS session.

Session Termination Phase Overview

A NetBIOS session is terminated normally when the user requests the session to be closed or when the session service detects the remote partner of the session has gracefully terminated the TCP connection. A NetBIOS session is abnormally terminated when the session service detects a loss of the connection. Connection loss can be detected with the keep-alive function of the session service or TCP, or on the failure of a SESSION MESSAGE send operation.

When a user requests to close a session, the service first attempts a graceful in-band close of the TCP connection. If the connection does not close within the SSN_CLOSE_TIMEOUT the TCP connection is aborted. No matter how the TCP connection is terminated, the NetBIOS session service always closes the NetBIOS session.

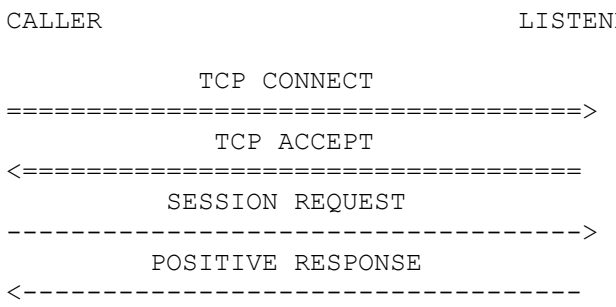
When the session service receives an indication from TCP that a connection close request has been received, the TCP connection and the NetBIOS session are immediately closed and the user is informed of the loss of the session. All data received up to the close indication should be delivered, if possible, to the session's user.

RFC-1001 Protocol Standard for a NetBIOS Service - Session Service

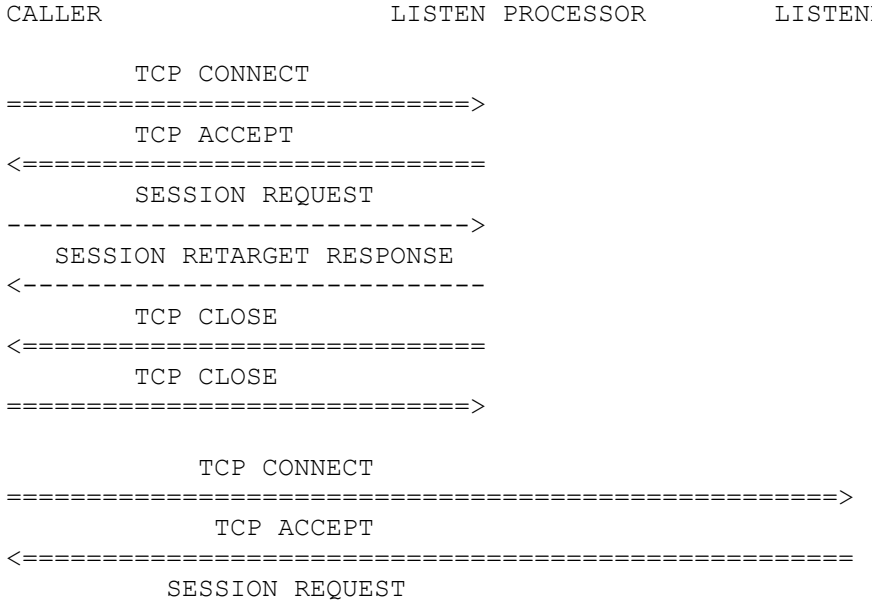
Session Establishment Phase

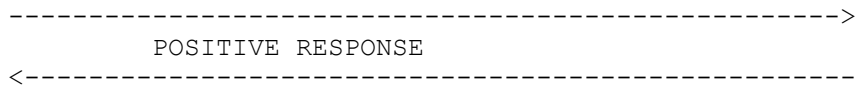
All the following diagrams assume a name query operation was successfully completed by the caller node for the listener's name.

This first diagram shows the sequence of network events used to successfully establish a session without retargetting by the listener. The TCP connection is first established with the well- known NetBIOS session service TCP port, SSN_SRVC_TCP_PORT. The caller then sends a SESSION REQUEST packet over the TCP connection requesting a session with the listener. The SESSION REQUEST contains the caller's name and the listener's name. The listener responds with a POSITIVE SESSION RESPONSE informing the caller this TCP connection is accepted as the connection for the data transfer phase of the session.

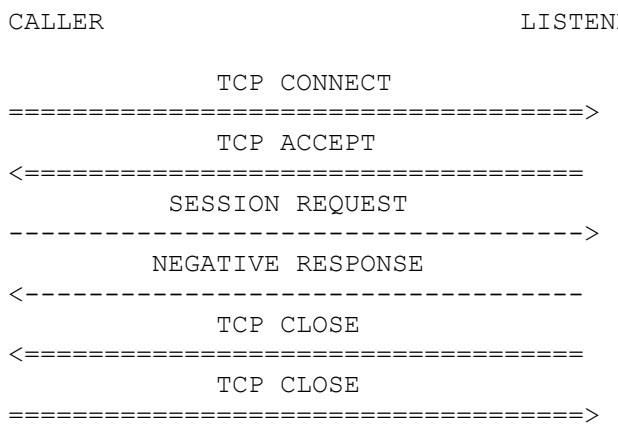


The second diagram shows the sequence of network events used to successfully establish a session when the listener does retargetting. The session establishment procedure is the same as with the first diagram up to the listener's response to the SESSION REQUEST. The listener, divided into two sections, the listen processor and the actual listener, sends a SESSION RETARGET RESPONSE to the caller. This response states the call is acceptable, but the data transfer TCP connection must be at the new IP address and TCP port. The caller then re-iterates the session establishment process anew with the new IP address and TCP port after the initial TCP connection is closed. The new listener then accepts this connection for the data transfer phase with a POSITIVE SESSION RESPONSE.

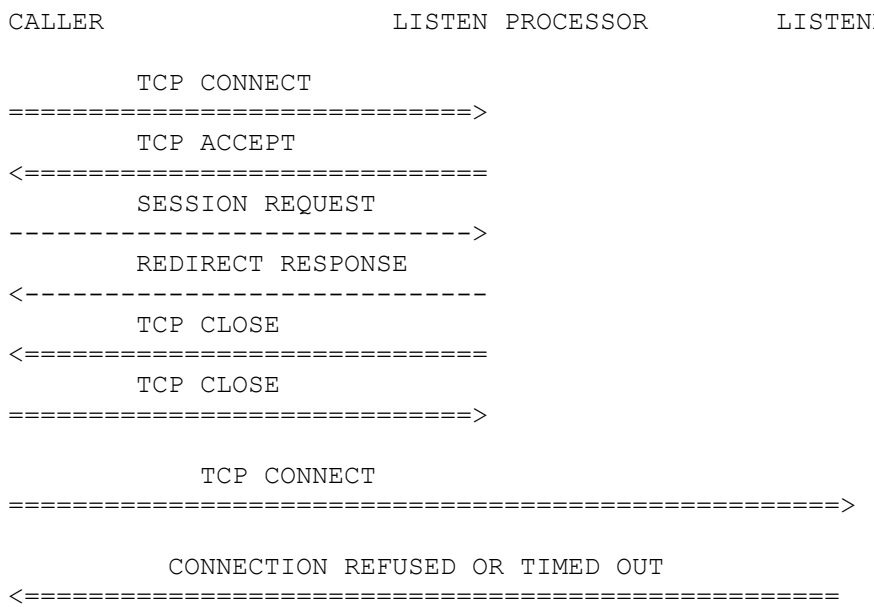




The third diagram is the sequence of network events for a rejected session request with the listener. This type of rejection could occur with either a non-retargetting listener or a retargetting listener. After the TCP connection is established at SSN_SRVC_TCP_PORT, the caller sends the SESSION REQUEST over the TCP connection. The listener does not have either a listen pending for the listener's name or the pending NetBIOS listen is specific to another caller's name. Consequently, the listener sends a NEGATIVE SESSION RESPONSE and closes the TCP connection.



The fourth diagram is the sequence of network events when session establishment fails with a retargetting listener. After being redirected, and after the initial TCP connection is closed the caller tries to establish a TCP connection with the new IP address and TCP port. The connection fails because either the port is unavailable or the target node is not active. The port unavailable race condition occurs if another caller has already acquired the TCP connection with the listener. For additional implementation suggestions, see Appendix B, "Retarget Algorithms".



Session Data Transfer Phase

Data Encapsulation
Session Keep-Alives

Data Encapsulation

NetBIOS messages are exchanged in the steady state. Messages are sent by prepending user data with message header and sending the header and the user data over the TCP connection. The receiver removes the header and delivers the NetBIOS data to the user.

Session Keep-Alives

In order to detect node failure or network partitioning, "session keep alive" packets are periodically sent in the steady state. A session keep alive packet is discarded by a peer node.

A session keep alive timer is maintained for each session. This timer is reset whenever any data is sent to, or received from, the session peer. When the timer expires, a NetBIOS session keep-alive packet is sent on the TCP connection. Sending the keep-alive packet forces data to flow on the TCP connection, thus indirectly causing TCP to detect whether the connection is still active.

Since many TCP implementations provide a parallel TCP "keep- alive" mechanism, the NetBIOS session keep-alive is made a configurable option. It is recommended that the NetBIOS keep- alive mechanism be used only in the absence of TCP keep-alive.

Note that unlike TCP keep alives, NetBIOS session keep alives do not require a response from the NetBIOS peer -- the fact that it was possible to send the NetBIOS session keep alive is sufficient indication that the peer, and the connection to it, are still active.

The only requirement for interoperability is that when a session keep alive packet is received, it should be discarded.

RFC-1001 Protocol Standard for a NetBIOS Service - Concepts and Methods

NetBIOS Datagram Service

RFC-1001 Protocol Standard for a NetBIOS Service - Datagram Service

Overview of NetBIOS Datagram Service

Every NetBIOS datagram has a named destination and source. To transmit a NetBIOS datagram, the datagram service must perform a name query operation to learn the IP address and the attributes of the destination NetBIOS name. (This information may be cached to avoid the overhead of name query on subsequent NetBIOS datagrams.)

NetBIOS datagrams are carried within UDP packets. If a NetBIOS datagram is larger than a single UDP packet, it may be fragmented into several UDP packets.

End-nodes may receive NetBIOS datagrams addressed to names not held by the receiving node. Such datagrams should be discarded. If the name is unique then a DATAGRAM ERROR packet is sent to the source of that NetBIOS datagram.

Unicast, Multicast, and Broadcast

NetBIOS datagrams may be unicast, multicast, or broadcast. A NetBIOS datagram addressed to a unique NetBIOS name is unicast. A NetBIOS datatgram addressed to a group NetBIOS name, whether there are zero, one, or more actual members, is multicast. A NetBIOS datagram sent using the NetBIOS "Send Broadcast Datagram" primitive is broadcast.

RFC-1001 Protocol Standard for a NetBIOS Service - Datagram Service

Fragmentation of NetBIOS Datagrams

When the header and data of a NetBIOS datagram exceeds the maximum amount of data allowed in a UDP packet, the NetBIOS datagram must be fragmented before transmission and reassembled upon receipt.

A NetBIOS Datagram is composed of the following protocol elements:

- IP header of 20 bytes (minimum)
- UDP header of 8 bytes
- NetBIOS Datagram Header of 14 bytes
- The NetBIOS Datagram data.

The NetBIOS Datagram data section is composed of 3 parts:

- Source NetBIOS name (255 bytes maximum)
- Destination NetBIOS name (255 bytes maximum)
- The NetBIOS user's data (maximum of 512 bytes)

The two name fields are in second level encoded format (see section 14.)

A maximum size NetBIOS datagram is 1064 bytes. The minimal maximum IP datagram size is 576 bytes. Consequently, a NetBIOS Datagram may not fit into a single IP datagram. This makes it necessary to permit the fragmentation of NetBIOS Datagrams.

On networks meeting or exceeding the minimum IP datagram length requirement of 576 octets, at most two NetBIOS datagram fragments will be generated. The protocols and packet formats accommodate fragmentation into three or more parts.

When a NetBIOS datagram is fragmented, the IP, UDP and NetBIOS Datagram headers are present in each fragment. The NetBIOS Datagram data section is split among resulting UDP datagrams. The data sections of NetBIOS datagram fragments do not overlap. The only fields of the NetBIOS Datagram header that would vary are the FLAGS and OFFSET fields.

The FIRST bit in the FLAGS field indicate whether the fragment is the first in a sequence of fragments. The MORE bit in the FLAGS field indicates whether other fragments follow.

The OFFSET field is the byte offset from the beginning of the NetBIOS datagram data section to the first byte of the data section in a fragment. It is 0 for the first fragment. For each subsequent fragment, OFFSET is the sum of the bytes in the NetBIOS data sections of all preceding fragments.

If the NetBIOS datagram was not fragmented:

- FIRST = TRUE
- MORE = FALSE
- OFFSET = 0

If the NetBIOS datagram was fragmented:

- First fragment:
- FIRST = TRUE
- MORE = TRUE
- OFFSET = 0

- Immediate fragments:
- FIRST = FALSE
- MORE = TRUE
- OFFSET = sum(NetBIOS data in prior fragments)

- Last fragment:
- FIRST = FALSE
- MORE = FALSE
- OFFSET = sum(NetBIOS data in prior fragments)

The relative position of intermediate fragments may be ascertained from OFFSET.

An NBDD must remember the destination name of the first fragment in order to relay the subsequent fragments of a single NetBIOS datagram. The name information can be associated with the subsequent fragments through the transaction ID, DGM_ID, and the SOURCE_IP, fields of the packet. This information can be purged by the NBDD after the last fragment has been processed or FRAGMENT_TO time has expired since the first fragment was received.

NetBIOS Datagrams by B Nodes

For NetBIOS datagrams with a named destination (i.e. non- broadcast), a B node performs a name discovery for the destination name before sending the datagram. (Name discovery may be bypassed if information from a previous discovery is held in a cache.) If the name type returned by name discovery is UNIQUE, the datagram is unicast to the sole owner of the name. If the name type is GROUP, the datagram is broadcast to the entire broadcast area using the destination IP address BROADCAST_ADDRESS.

A receiving node always filters datagrams based on the destination name. If the destination name is not owned by the node or if no RECEIVE DATAGRAM user operations are pending for the name, then the datagram is discarded. For datagrams with a UNIQUE name destination, if the name is not owned by the node then the receiving node sends a DATAGRAM ERROR packet. The error packet originates from the DGM_SRVC_UDP_PORT and is addressed to the SOURCE_IP and SOURCE_PORT from the bad datagram. The receiving node quietly discards datagrams with a GROUP name destination if the name is not owned by the node.

Since broadcast NetBIOS datagrams do not have a named destination, the B node sends the DATAGRAM SERVICE packet(s) to the entire broadcast area using the destination IP address BROADCAST_ADDRESS. In order for the receiving nodes to distinguish this datagram as a broadcast NetBIOS datagram, the NetBIOS name used as the destination name is '*' (hexadecimal 2A) followed by 15 bytes of hexidecimal 00. The NetBIOS scope identifier is appended to the name before it is converted into second-level encoding. For example, if the scope identifier is "NETBIOS.SCOPE" then the first-level encoded name would be:

CKAAAAAAAAAAAAAAAAAAAAAAAAAAAA.NETBIOS.SCOPE

According to [2], a user may not provide a NetBIOS name beginning with "*".

For each node in the broadcast area that receives the NetBIOS broadcast datagram, if any RECEIVE BROADCAST DATAGRAM user operations are pending then the data from the NetBIOS datagram is replicated and delivered to each. If no such operations are pending then the node silently discards the datagram.

NetBIOS Datagrams by P and M Nodes

P and M nodes do not use IP broadcast to distribute NetBIOS datagrams.

Like B nodes, P and M nodes must perform a name discovery or use cached information to learn whether a destination name is a group or a unique name.

Datagrams to unique names are unicast directly to the destination by P and M nodes, exactly as they are by B nodes.

Datagrams to group names and NetBIOS broadcast datagrams are unicast to the NBDD. The NBDD then relays the datagrams to each of the nodes specified by the destination name.

An NBDD may not be capable of sending a NetBIOS datagram to a particular NetBIOS name, including the broadcast NetBIOS name ("*") defined above. A query mechanism is available to the end- node to determine if a NBDD will be able to relay a datagram to a given name. Before a datagram, or its fragments, are sent to the NBDD the P or M node may send a DATAGRAM QUERY REQUEST packet to the NBDD with the DESTINATION_NAME from the DATAGRAM SERVICE packet(s). The NBDD will respond with a DATAGRAM POSITIVE QUERY RESPONSE if it will relay datagrams to the specified destination name. After a positive response the end-node unicasts the datagram to the NBDD. If the NBDD will not be able to relay a datagram to the destination name specified in the query, a DATAGRAM NEGATIVE QUERY RESPONSE packet is returned. If the NBDD can not distribute a datagram, the end-node then has the option of getting the name's owner list from the NBNS and sending the datagram directly to each of the owners.

An NBDD must be able to respond to DATAGRAM QUERY REQUEST packets. The response may always be positive. However, the usage or implementation of the query mechanism by a P or M node is optional. An implementation may always unicast the NetBIOS datagram to the NBDD without asking if it will be relayed. Except for the datagram query facility described above, an NBDD provides no feedback to indicate whether it forwarded a datagram.

Node Configuration Parameters

- B NODES:
- Node's permanent unique name
- Whether IGMP is in use
- Broadcast IP address to use
- Whether NetBIOS session keep-alives are needed
- Usable UDP data field length (to control fragmentation)
- P NODES:
- Node's permanent unique name
- IP address of NBNS
- IP address of NBDD
- Whether NetBIOS session keep-alives are needed
- Usable UDP data field length (to control fragmentation)
- M NODES:
- Node's permanent unique name
- Whether IGMP is in use
- Broadcast IP address to use
- IP address of NBNS
- IP address of NBDD
- Whether NetBIOS session keep-alives are needed
- Usable UDP data field length (to control fragmentation)

Minimal Conformance

To ensure multi-vendor interoperability, a minimally conforming implementation based on this specification must observe the following rules:

a) A node designed to work only in a broadcast area must conform to the B node specification.

b) A node designed to work only in an internet must conform to the P node specification.

Appendix A

Integration With Internet Group Multicasting

This appendix contains supporting technical discussions. It is not an integral part of the NetBIOS-over-TCP specification.

The Netbios-over-TCP system described in this RFC may be easily integrated with the Internet Group Multicast system now being developed for the internet.

In the main body of the RFC, the notion of a broadcast area was considered to be a single MAC-bridged "B-LAN". However, the protocols defined will operate over an extended broadcast area resulting from the creation of a permanent Internet Multicast Group.

Each separate broadcast area would be based on a separate permanent Internet Multicast Group. This multicast group address would be used by B and M nodes as their BROADCAST_ADDRESS.

In order to base the broadcast area on a multicast group certain additional procedures are required and certain constraints must be met.

Additional Protocol Required in B and M Nodes

All B or M nodes operating on an IGMP based broadcast area must have IGMP support in their IP layer software. These nodes must perform an IGMP join operation to enter the IGMP group before engaging in NetBIOS activity.

Constraints

Broadcast Areas may overlap. For this reason, end-nodes must be careful to examine the NetBIOS scope identifiers in all received broadcast packets.

The NetBIOS broadcast protocols were designed for a network that exhibits a low average transit time and low rate of packet loss. An IGMP based broadcast area must exhibit these characteristics. In practice this will tend to constrain IGMP broadcast areas to a campus of networks interconnected by high-speed routers and inter-router links. It is unlikely that transcontinental broadcast areas would exhibit the required characteristics.

Appendix B

Implementation Considerations

This appendix contains supporting technical discussions. It is not an integral part of the NetBIOS-over-TCP specification.

RFC-1001 Protocol Standard for a NetBIOS Service - Implementation Considerations

Implementation Models

On any participating system, there must be some sort of NetBIOS Service to coordinate access by NetBIOS applications on that system.

To analyze the impact of the NetBIOS-over-TCP architecture, we use the following three models of how a NetBIOS service might be implemented:

1. Combined Service and Application Model
   The NetBIOS service and application are both contained within a single process. No interprocess communication is assumed within the system; all communication is over the network. If multiple applications require concurrent access to the NetBIOS service, they must be folded into this monolithic process.
2. Common Kernel Element Model
   The NetBIOS Service is part of the operating system (perhaps as a device driver or a front-end processor). The NetBIOS applications are normal operating system application processes. The common element NetBIOS service contains all the information, such as the name and listen tables, required to co-ordinate the activities of the applications.
3. Non-Kernel Common Element Model
   The NetBIOS Service is implemented as an operating system application process. The NetBIOS applications are other operating system application processes. The service and the applications exchange data via operating system interprocess communication. In a multi-processor (e.g. network) operating system, each module may reside on a different cpu. The NetBIOS service process contains all the shared information required to coordinate the activities of the NetBIOS applications. The applications may still require a subroutine library to facilitate access to the NetBIOS service.

For any of the implementation models, the TCP/IP service can be located in the operating system or split among the NetBIOS applications and the NetBIOS service processes.

Model Independent Considerations

The NetBIOS name service associates a NetBIOS name with a host. The NetBIOS session service further binds the name to a specific TCP port for the duration of the session.

The name service does not need to be informed of every Listen initiation and completion. Since the names are not bound to any TCP port in the name service, the session service may use a different tcp port for each session established with the same local name.

The TCP port used for the data transfer phase of a NetBIOS session can be globally well-known, locally well-known, or ephemeral. The choice is a local implementation issue. The RETARGET mechanism allows the binding of the NetBIOS session to a TCP connection to any TCP port, even to another IP node.

An implementation may use the session service's globally well- known TCP port for the data transfer phase of the session by not using the RETARGET mechanism and, rather, accepting the session on the initial TCP connection. This is permissible because the caller always uses an ephemeral TCP port.

The complexity of the called end RETARGET mechanism is only required if a particular implementation needs it. For many real system environments, such as an in-kernel NetBIOS service implementation, it will not be necessary to retarget incoming calls. Rather, all NetBIOS sessions may be multiplexed through the single, well-known, NetBIOS session service port. These implementations will not be burdened by the complexity of the RETARGET mechanism, nor will their callers be required to jump through the retargetting hoops.

Nevertheless, all callers must be ready to process all possible SESSION RETARGET RESPONSEs.

Service Operation for Each Model

It is possible to construct a NetBIOS service based on this specification for each of the above defined implementation models.

For the common kernel element model, all the NetBIOS services, name, datagram, and session, are simple. All the information is contained within a single entity and can therefore be accessed or modified easily. A single port or multiple ports for the NetBIOS sessions can be used without adding any significant complexity to the session establishment procedure. The only penalty is the amount of overhead incurred to get the NetBIOS messages and operation requests/responses through the user and operating system boundary.

The combined service and application model is very similar to the common kernel element model in terms of its requirements on the NetBIOS service. The major difficulty is the internal coordination of the multiple NetBIOS service and application processes existing in a system of this type.

The NetBIOS name, datagram and session protocols assume that the entities at the end-points have full control of the various well- known TCP and UDP ports. If an implementation has multiple NetBIOS service entities, as would be the case with, for example, multiple applications each linked into a NetBIOS library, then that implementation must impose some internal coordination. Alternatively, use of the NetBIOS ports could be periodically assigned to one application or another.

For the typical non-kernel common element mode implementation, three permanent system-wide NetBIOS service processes would exist:

- The name server
- the datagram server
- and session server

Each server would listen for requests from the network on a UDP or TCP well-known port. Each application would have a small piece of the NetBIOS service built-in, possibly a library. Each application's NetBIOS support library would need to send a message to the particular server to request an operation, such as add name or send a datagram or set-up a listen.

The non-kernel common element model does not require a TCP connection be passed between the two processes, session server and application. The RETARGET operation for an active NetBIOS Listen could be used by the session server to redirect the session to another TCP connection on a port allocated and owned by the application's NetBIOS support library. The application with either a built-in or a kernel-based TCP/IP service could then accept the RETARGETed connection request and process it independently of the session server.

On Unix(tm) or POSIX(tm), the NetBIOS session server could create sub-processes for incoming connections. The open sessions would be passed through "fork" and "exec" to the child as an open file descriptor. This approach is very limited, however. A pre- existing process could not receive an incoming call. And all call-ed processes would have to be sub-processes of the session server.

Casual and Restricted NetBIOS Applications

Because NetBIOS was designed to operate in the open system environment of the typical personal computer, it does not have the concept of privileged or unprivileged applications. In many multi- user or multi-tasking operating systems applications are assigned privilege capabilities. These capabilities limit the applications ability to acquire and use system resources. For these systems it is important to allow casual applications, those with limited system privileges, and privileged applications, those with 'super-user' capabilities but access to them and their required resources is restricted, to access NetBIOS services. It is also important to allow a systems administrator to restrict certain NetBIOS resources to a particular NetBIOS application. For example, a file server based on the NetBIOS services should be able to have names and TCP ports for sessions only it can use.

A NetBIOS application needs at least two local resources to communicate with another NetBIOS application, a NetBIOS name for itself and, typically, a session. A NetBIOS service cannot require that NetBIOS applications directly use privileged system resources. For example, many systems require privilege to use TCP and UDP ports with numbers less than 1024. This RFC requires reserved ports for the name and session servers of a NetBIOS service implementation. It does not require the application to have direct access these reserved ports.

For the name service, the manager of the local name table must have access to the NetBIOS name service's reserved UDP port. It needs to listen for name service UDP packets to defend and define its local names to the network. However, this manager need not be a part of a user application in a system environment which has privilege restrictions on reserved ports.

The internal name server can require certain privileges to add, delete, or use a certain name, if an implementer wants the restriction. This restriction is independent of the operation of the NetBIOS service protocols and would not necessarily prevent the interoperation of that implementation with another implementation.

The session server is required to own a reserved TCP port for session establishment. However, the ultimate TCP connection used to transmit and receive data does not have to be through that reserved port. The RETARGET procedure the NetBIOS session to be shifted to another TCP connection, possibly through a different port at the called end. This port can be an unprivileged resource, with a value greater than 1023. This facilitates casual applications.

Alternately, the RETARGET mechanism allows the TCP port used for data transmission and reception to be a reserved port. Consequently, an application wishing to have access to its ports maintained by the system administrator can use these restricted TCP ports. This facilitates privileged applications.

A particular implementation may wish to require further special privileges for session establishment, these could be associated with internal information. It does not have to be based solely on TCP port allocation. For example, a given NetBIOS name may only be used for sessions by applications with a certain system privilege level.

The decision to use reserved or unreserved ports or add any additional name registration and usage authorization is a purely local implementation decision. It is not required by the NetBIOS protocols specified in the RFC.

TCP Versus Session Keep-Alives

The KEEP-ALIVE is a protocol element used to validate the existence of a connection. A packet is sent to the remote connection partner to solicit a response which shows the connection is still functioning. TCP KEEP-ALIVES are used at the TCP level on TCP connections while session KEEP-ALIVES are used on NetBIOS sessions. These protocol operations are always transparent to the connection user. The user will only find out about a KEEP-ALIVE operation if it fails, therefore, if the connection is lost.

The NetBIOS specification[2] requires the NetBIOS service to inform the session user if a session is lost when it is in a passive or active state. Therefore,if the session user is only waiting for a receive operation and the session is dropped the NetBIOS service must inform the session user. It cannot wait for a session send operation before it informs the user of the loss of the connection.

This requirement stems from the management of scarce or volatile resources by a NetBIOS application. If a particular user terminates a session with a server application by destroying the client application or the NetBIOS service without a NetBIOS Hang Up, the server application will want to clean-up or free allocated resources. This server application if it only receives and then sends a response requires the notification of the session abort in the passive state.

The standard definition of a TCP service cannot detect loss of a connection when it is in a passive state, waiting for a packet to arrive. Some TCP implementations have added a KEEP-ALIVE operation which is interoperable with implementations without this feature. These implementations send a packet with an invalid sequence number to the connection partner. The partner, by specification, must respond with a packet showing the correct sequence number of the connection. If no response is received from the remote partner within a certain time interval then the TCP service assumes the connection is lost.

Since many TCP implementations do not have this KEEP-ALIVE function an optional NetBIOS KEEP-ALIVE operation has been added to the NetBIOS session protocols. The NetBIOS KEEP-ALIVE uses the properties of TCP to solicit a response from the remote connection partner. A NetBIOS session message called KEEP-ALIVE is sent to the remote partner. Since this results in TCP sending an IP packet to the remote partner, the TCP connection is active. TCP will discover if the TCP connection is lost if the remote TCP partner does not acknowledge the IP packet. Therefore, the NetBIOS session service does not send a response to a session KEEP ALIVE message. It just throws it away. The NetBIOS session service that transmits the KEEP ALIVE is informed only of the failure of the TCP connection. It does not wait for a specific response message.

A particular NetBIOS implementation should use KEEP-ALIVES if it is concerned with maintaining compatibility with the NetBIOS interface specification[2]. Compatibility is especially important if the implementation wishes to support existing NetBIOS applications, which typically require the session loss detection on their servers, or future applications which were developed for implementations with session loss detection.

Retarget Algorithms

This section contains 2 suggestions for RETARGET algorithms. They are called the "straight" and "stack" methods. The algorithm in the body of the RFC uses the straight method. Implementation of either algorithm must take into account the Session establishment maximum retry count. The retry count is the maximum number of TCP connect operations allowed before a failure is reported.

The straight method forces the session establishment procedure to begin a retry after a retargetting failure with the initial node returned from the name discovery procedure. A retargetting failure is when a TCP connection attempt fails because of a time- out or a NEGATIVE SESSION RESPONSE is received with an error code specifying NOT LISTENING ON CALLED NAME. If any other failure occurs the session establishment procedure should retry from the call to the name discovery procedure.

A minimum of 2 retries, either from a retargetting or a name discovery failure. This will give the session service a chance to re-establish a NetBIOS Listen or, more importantly, allow the NetBIOS scope, local name service or the NBNS, to re-learn the correct IP address of a NetBIOS name.

The stack method operates similarly to the straight method. However, instead of retrying at the initial node returned by the name discovery procedure, it restarts with the IP address of the last node which sent a SESSION RETARGET RESPONSE prior to the retargetting failure. To limit the stack method, any one host can only be tried a maximum of 2 times.

NBDD Service

If the NBDD does not forward datagrams then don't provide Group and Broadcast NetBIOS datagram services to the NetBIOS user. Therefore, ignore the implementation of the query request and, when get a negative response, acquiring the membership list of IP addresses and sending the datagram as a unicast to each member.

Use of NetBIOS Datagrams

Certain existing NetBIOS applications use NetBIOS datagrams as a foundation for their own connection-oriented protocols. This can cause excessive NetBIOS name query activity and place a substantial burden on the network, server nodes, and other end- nodes. It is recommended that this practice be avoided in new applications.

References

[1] "Protocol Standard For a NetBIOS Service on a TCP/UDP Transport: Detailed Specifications", RFC 1002, March 1987.

[2] IBM Corp., "IBM PC Network Technical Reference Manual", No. 6322916, First Edition, September 1984.

[3] J. Postel (Ed.), "Transmission Control Protocol", RFC 793, September 1981.

[4] MIL-STD-1778

[5] J. Postel, "User Datagram Protocol", RFC 768, 28 August 1980.

[6] J. Reynolds, J. Postel, "Assigned Numbers", RFC 990, November 1986.

[7] J. Postel, "Internet Protocol", RFC 791, September 1981.

[8] J. Mogul, "Internet Subnets", RFC 950, October 1984

[9] J. Mogul, "Broadcasting Internet Datagrams in the Presence of Subnets", RFC 922, October 1984.

[10] J. Mogul, "Broadcasting Internet Datagrams", RFC 919, October 1984.

[11] P. Mockapetris, "Domain Names - Concepts and Facilities", RFC 882, November 1983.

[12] P. Mockapetris, "Domain Names - Implementation and Specification", RFC 883, November 1983.

[13] P. Mockapetris, "Domain System Changes and Observations", RFC 973, January 1986.

[14] C. Partridge, "Mail Routing and the Domain System", RFC 974, January 1986.

[15] S. Deering, D. Cheriton, "Host Groups: A Multicast Extension to the Internet Protocol", RFC 966, December 1985.

[16] S. Deering, "Host Extensions for IP Multicasting", RFC 988, July 1986.

RFC-1002 Protocol Standard for a NetBIOS Service on a TCP/UDP Transport Detailed Specifications

March 1987

Abstract

This RFC defines an elective standard protocol to support NetBIOS services in a TCP/IP environment. Both local network and internet operation are supported. Various node types are defined to accommodate local and internet topologies and to allow operation with or without the use of IP broadcast.

This RFC gives the detailed specifications of the NetBIOS-over-TCP packets, protocols, and defined constants and variables. A more general overview is found in a companion RFC, "Protocol Standard For a NetBIOS Service on a TCP/UDP Transport: Concepts and Methods" [RFC-1001].

RFC-1002 Protocol Standard for a NetBIOS Service - Detailed Specifications

Name Service Packets

Session Service Packets

RFC-1002 Protocol Standard for a NetBIOS Service - Detailed Specifications

Datagram Service Packets

RFC-1002 Protocol Standard for a NetBIOS Service - Detailed Specifications

Name Service Protocols

Session Service Protocols

RFC-1002 Protocol Standard for a NetBIOS Service - Detailed Specifications

NetBIOS Datagram Service Protocols

Status of This Memo

This RFC specifies an elective standard for the DARPA Internet community. Since this topic is new to the Internet community, discussions and suggestions are specifically requested.

Please send written comments to:

Karl Auerbach
Epilogue Technology Corporation
P.O. Box 5432
Redwood City, CA 94063

Please send online comments to:

Avnish Aggarwal
Internet: mtxinu!excelan!avnish@ucbvax.berkeley.edu
Usenet: ucbvax!mtxinu!excelan!avnish

Acknowledgements

This RFC has been developed under the auspices of the Internet Activities Board.

The following individuals have contributed to the development of this RFC:

Avnish Aggarwal	Arvind Agrawal	Lorenzo Aguilar
Geoffrey Arnold	Karl Auerbach	K. Ramesh Babu
Keith Ball	Amatzia Ben-Artzi	Vint Cerf
Richard Cherry	David Crocker	Steve Deering
Greg Ennis	Steve Holmgren	Jay Israel
David Kaufman	Lee LaBarre	James Lau
Dan Lynch	Gaylord Miyata	David Stevens
Steve Thomas	Ishan Wu	

The system proposed by this RFC does not reflect any existing Netbios-over-TCP implementation. However, the design incorporates considerable knowledge obtained from prior implementations. Special thanks goes to the following organizations which have provided this invaluable information:

CMC/Syros Excelan Sytek Ungermann-Bass

RFC-1002 Protocol Standard for a NetBIOS Service - Detailed Specifications

Introduction

This RFC contains the detailed packet formats and protocol specifications for NetBIOS-over-TCP. This RFC is a companion to RFC 1001, "Protocol Standard For a NetBIOS Service on a TCP/UDP Transport: Concepts and Methods" [RFC-1001].

Packet Descriptions

Bit and byte ordering are defined by the most recent version of "Assigned Numbers" [RFC-1060].

Name Format

The NetBIOS name representation in all NetBIOS packets (for NAME, SESSION, and DATAGRAM services) is defined in the Domain Name Service RFC 883[RFC-833] as "compressed" name messages. This format is called "second-level encoding" in the section entitled "Representation of NetBIOS Names" in the Concepts and Methods document.

For ease of description, the first two paragraphs from page 31, the section titled "Domain name representation and compression", of RFC 883 are replicated here:

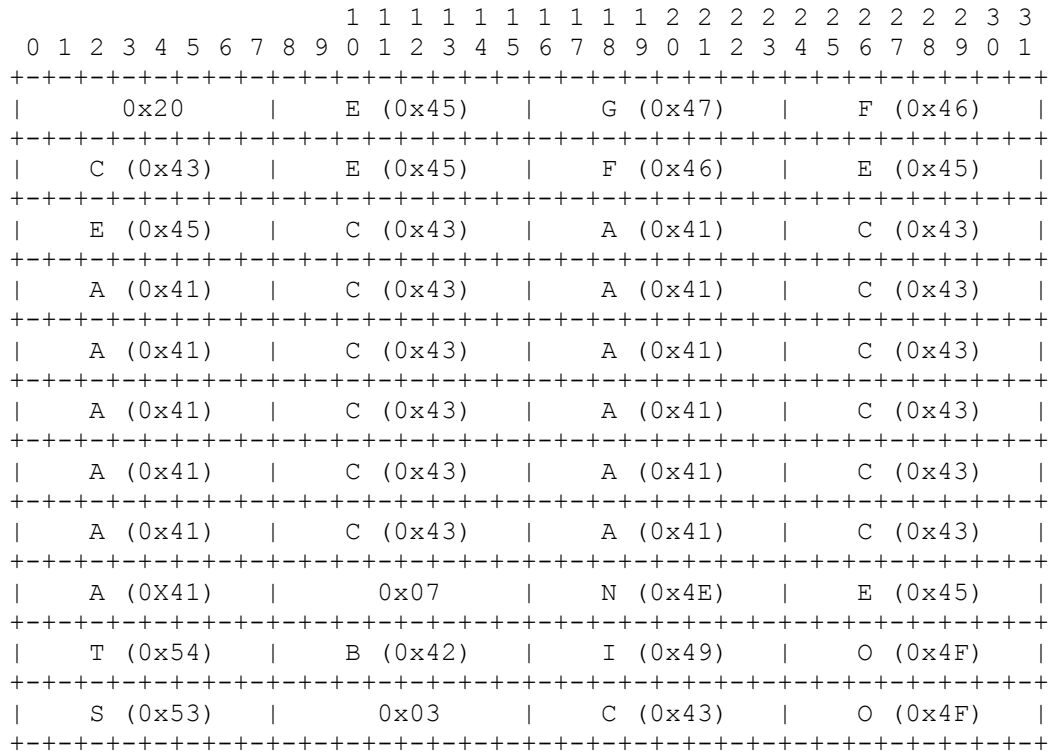
> Domain names messages are expressed in terms of a sequence of labels. Each label is represented as a one octet length field followed by that number of octets. Since every domain name ends with the null label of the root, a compressed domain name is terminated by a length byte of zero. The high order two bits of the length field must be zero, and the remaining six bits of the length field limit the label to 63 octets or less.
>
> To simplify implementations, the total length of label octets and label length octets that make up a domain name is restricted to 255 octets or less.

The following is the uncompressed representation of the NetBIOS name "FRED ", which is the 4 ASCII characters, F, R, E, D, followed by 12 space characters (0x20). This name has the SCOPE_ID: "NETBIOS.COM"

> EGFCEFEECACACACACACACACACACACACA.NETBIOS.COM

This uncompressed representation of names is called "first-level encoding" in the section entitled "Representation of NetBIOS Names" in the Concepts and Methods document.

The following is a pictographic representation of the compressed representation of the previous uncompressed Domain Name representation.



```
|    M (0x4D)    |       0x00      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Each section of a domain name is called a label [7 (page 31)]. A label can be a maximum of 63 bytes. The first byte of a label in compressed representation is the number of bytes in the label. For the above example, the first 0x20 is the number of bytes in the left-most label, EGFCEFEECACACACACACACACACACACACA, of the domain name. The bytes following the label length count are the characters of the label. The following labels are in sequence after the first label, which is the encoded NetBIOS name, until a zero (0x00) length count. The zero length count represents the root label, which is always null.

A label length count is actually a 6-bit field in the label length field. The most significant 2 bits of the field, bits 7 and 6, are flags allowing an escape from the above compressed representation. If bits 7 and 6 are both set (11), the following 14 bits are an offset pointer into the full message to the actual label string from another domain name that belongs in this name. This label pointer allows for a further compression of a domain name in a packet.

NetBIOS implementations can only use label string pointers in Name Service packets. They cannot be used in Session or Datagram Service packets.

The other two possible values for bits 7 and 6 (01 and 10) of a label length field are reserved for future use by RFC 883[2 (page 32)].

Note that the first octet of a compressed name must contain one of the following bit patterns. (An "x" indicates a bit whose value may be either 0 or 1.):

```
00100000    - Netbios name, length must be 32 (decimal)
11xxxxxx    - Label string pointer
10xxxxxx    - Reserved
01xxxxxx    - Reserved
```

RFC-1002 Protocol Standard for a NetBIOS Service - Packet Descriptions

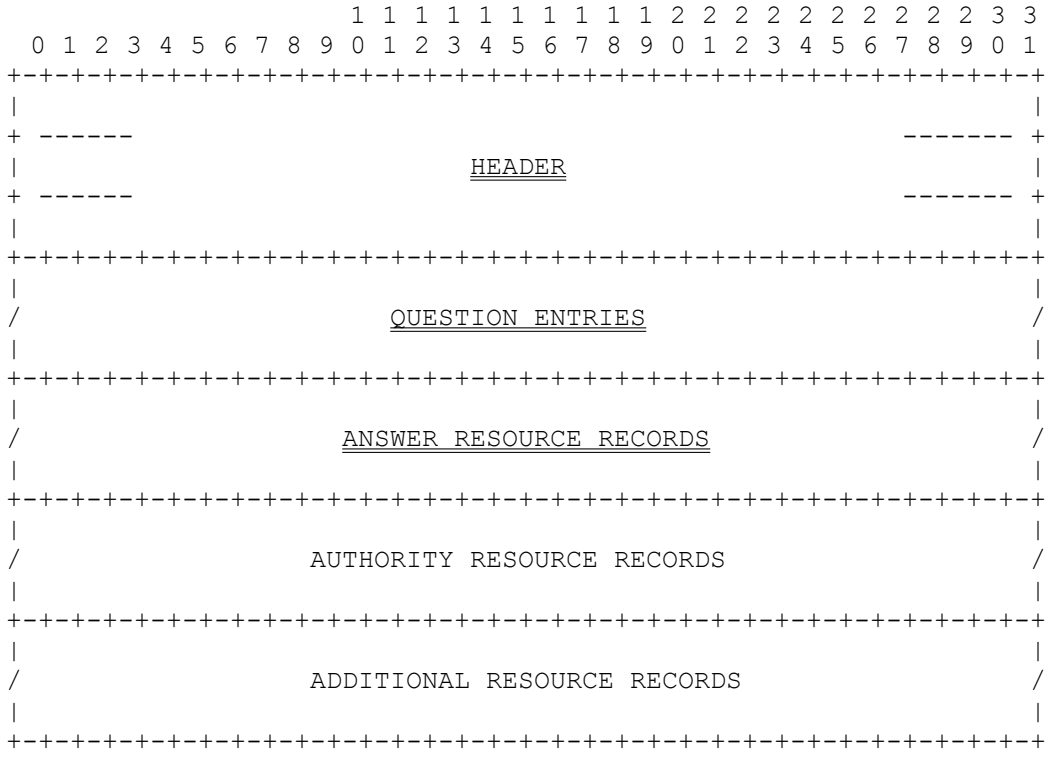
Name Service Packets

RFC-1002 Protocol Standard for a NetBIOS Service - Name Service Packets

General Format Of Name Service Packets

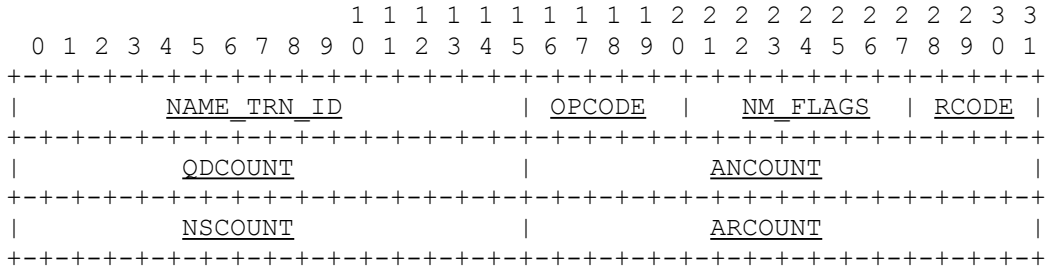
The NetBIOS Name Service packets follow the packet structure defined in the Domain Name Service (DNS) RFC 883 [7 (pg 26-31)]. The structures are compatible with the existing DNS packet formats, however, additional types and codes have been added to work with NetBIOS.

If Name Service packets are sent over a TCP connection they are preceded by a 16 bit unsigned integer representing the length of the Name Service packet.



RFC-1002 Protocol Standard for a NetBIOS Service - Name Service Packets

Header

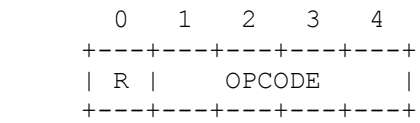


Field	Description
NAME_TRN_ID	Transaction ID for Name Service Transaction. Requestor places a unique value for each active transaction. Responder puts NAME_TRN_ID value from request packet in response packet.
OPCODE	Packet type code, see table below.
NM_FLAGS	Flags for operation, see table below.
RCODE	Result codes of request. Table of RCODE values for each response packet below.
QDCOUNT	Unsigned 16 bit integer specifying the number of entries in the question section of a Name

Service packet. Always zero (0) for responses. Must be non-zero for all NetBIOS Name requests.

ANCOUNT	Unsigned 16 bit integer specifying the number of resource records in the answer section of a Name Service packet.
NSCOUNT	Unsigned 16 bit integer specifying the number of resource records in the authority section of a Name Service packet.
ARCOUNT	Unsigned 16 bit integer specifying the number of resource records in the additional records section of a Name Service packet.

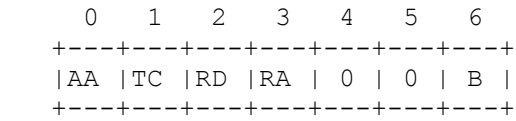
The OPCODE field is defined as:



Symbol	Bit(s)	Description
OPCODE	1-4	Operation specifier: 0 = query 5 = registration 6 = release 7 = WACK 8 = refresh

R	0	RESPONSE flag: if bit == 0 then request packet if bit == 1 then response packet.

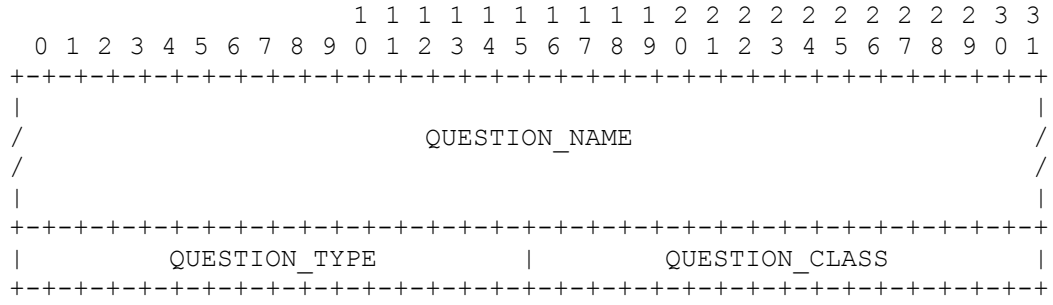
The NM_FLAGS field is defined as:



Symbol	Bit(s)	Description
B	6	Broadcast Flag. = 1: packet was broadcast or multicast = 0: unicast
RA	3	Recursion Available Flag. Only valid in responses from a NetBIOS Name Server -- must be zero in all other responses. If one (1) then the NBNS supports recursive query, registration, and release. If zero (0) then the end-node must iterate for query and challenge for registration.
RD	2	Recursion Desired Flag. May only be set on a request to a NetBIOS Name Server. The NBNS will copy its state into the response packet. If one (1) the NBNS will iterate on the query, registration, or release.
TC	1	Truncation Flag. Set if this message was truncated because the datagram carrying it would be greater than 576 bytes in length. Use TCP to get the information from the NetBIOS Name Server.
AA	0	Authoritative Answer flag. Must be zero (0) if R flag of OPCODE is zero (0). If R flag is one (1) then if AA is one (1) then the node responding is an authority for the domain name. End nodes responding to queries always set this bit in responses.

RFC-1002 Protocol Standard for a NetBIOS Service - Name Service Packets

Question Section



Field	Description
QUESTION_NAME	The compressed name representation of the NetBIOS name for the request.
QUESTION_TYPE	The type of request. The values for this field are specified for each request.
QUESTION_CLASS	The class of the request. The values for this field are specified for each request.

QUESTION_TYPE is defined as:

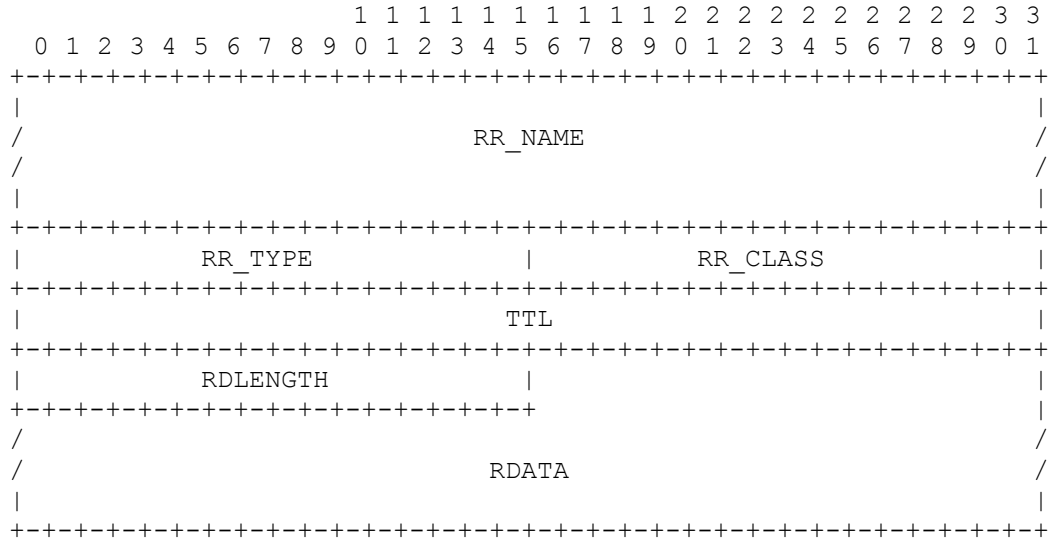
Symbol	Value	Description:
NB	0x0020	NetBIOS general Name Service Resource Record
NBSTAT	0x0021	NetBIOS NODE STATUS Resource Record (See NODE STATUS REQUEST)

QUESTION_CLASS is defined as:

Symbol	Value	Description:
IN	0x0001	Internet class

RFC-1002 Protocol Standard for a NetBIOS Service - Name Service Packets

Resource Record



Field	Description
RR_NAME	The compressed name representation of the NetBIOS name corresponding to this resource record.
RR_TYPE	Resource record type code
RR_CLASS	Resource record class code
TTL	The Time To Live of a the resource record's name.
RDLENGTH	Unsigned 16 bit integer that specifies the number of bytes in the RDATA field.
RDATA	RR_CLASS and RR_TYPE dependent field. Contains the resource information for the NetBIOS name.

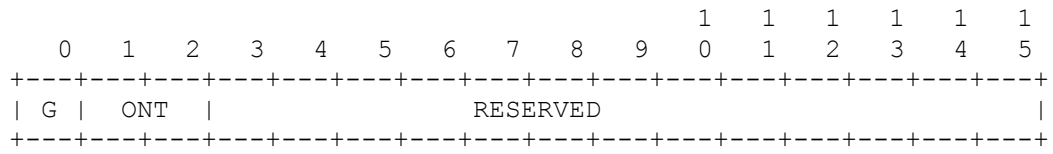
RESOURCE RECORD RR_TYPE field definitions:

Symbol	Value	Description:
A	0x0001	IP address Resource Record (See REDIRECT NAME QUERY RESPONSE)
NS	0x0002	Name Server Resource Record (See REDIRECT NAME QUERY RESPONSE)
NULL	0x000A	NULL Resource Record (See WAIT FOR ACKNOWLEDGEMENT RESPONSE)
NB	0x0020	NetBIOS general Name Service Resource Record (See NB_FLAGS and NB_ADDRESS, below)
NBSTAT	0x0021	NetBIOS NODE STATUS Resource Record (See NODE STATUS RESPONSE)

RESOURCE RECORD RR_CLASS field definitions:

Symbol	Value	Description:
IN	0x0001	Internet class

NB_FLAGS field of the RESOURCE RECORD RDATA field for RR_TYPE of "NB":

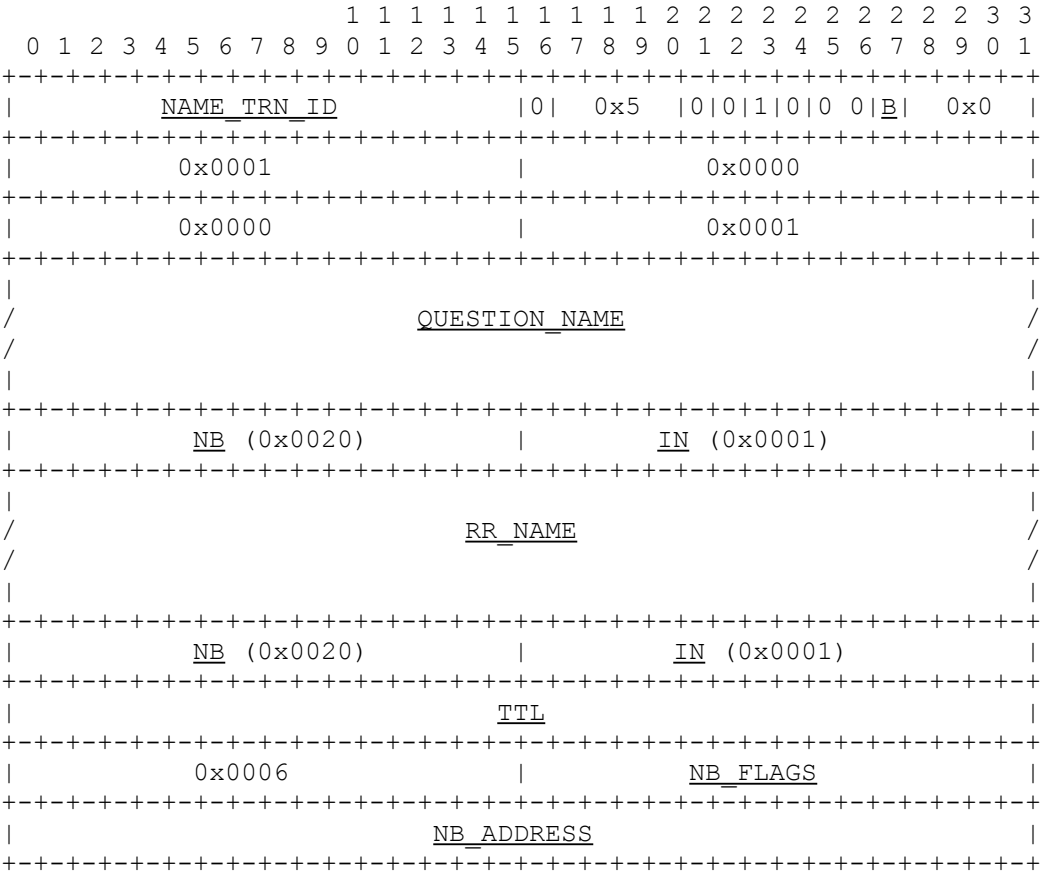


Symbol	Bit(s)	Description:
RESERVED	3-15	Reserved for future use. Must be zero (0).
ONT	1,2	Owner Node Type: 00 = B node 01 = P node 10 = M node 11 = Reserved for future use For registration requests this is the claimant's type. For responses this is the actual owner's type.
G	0	Group Name Flag. If one (1) then the RR_NAME is a GROUP NetBIOS name. If zero (0) then the RR_NAME is a UNIQUE NetBIOS name.

The NB_ADDRESS field of the RESOURCE RECORD RDATA field for RR_TYPE of "NB" is the IP address of the name's owner.

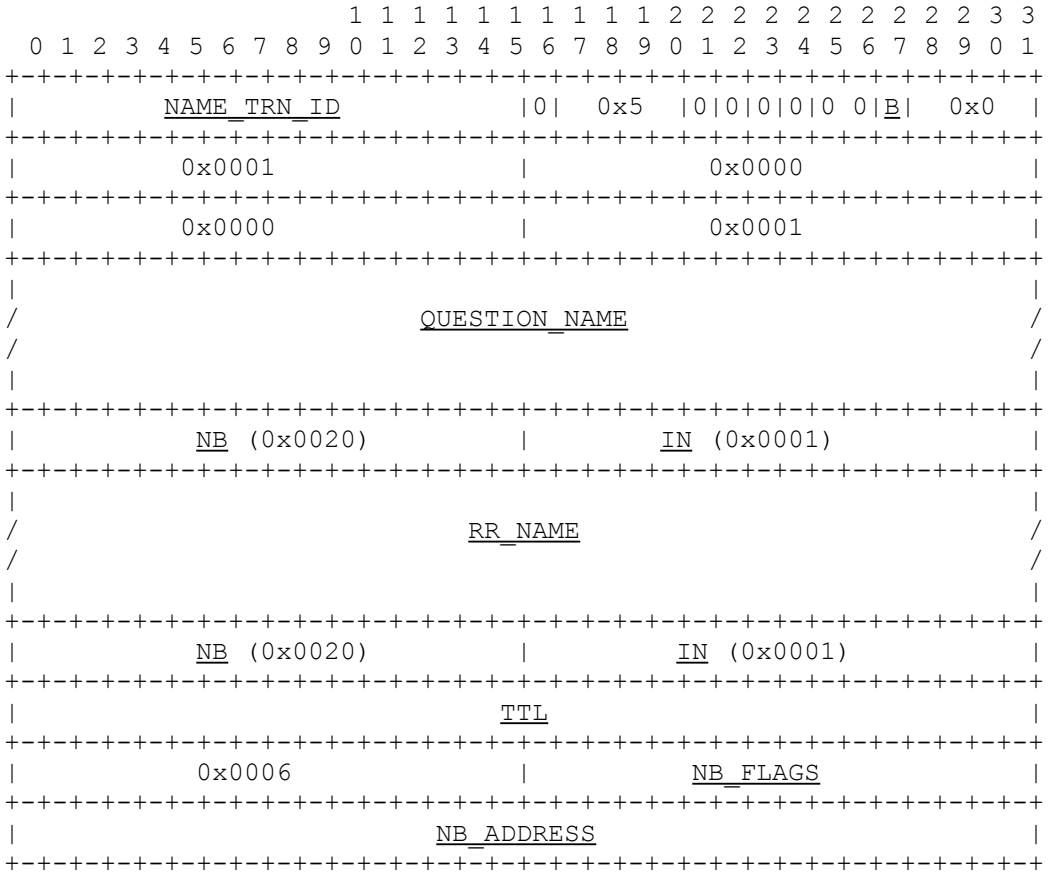
RFC-1002 Protocol Standard for a NetBIOS Service - Name Service Packets

Name Registration Request

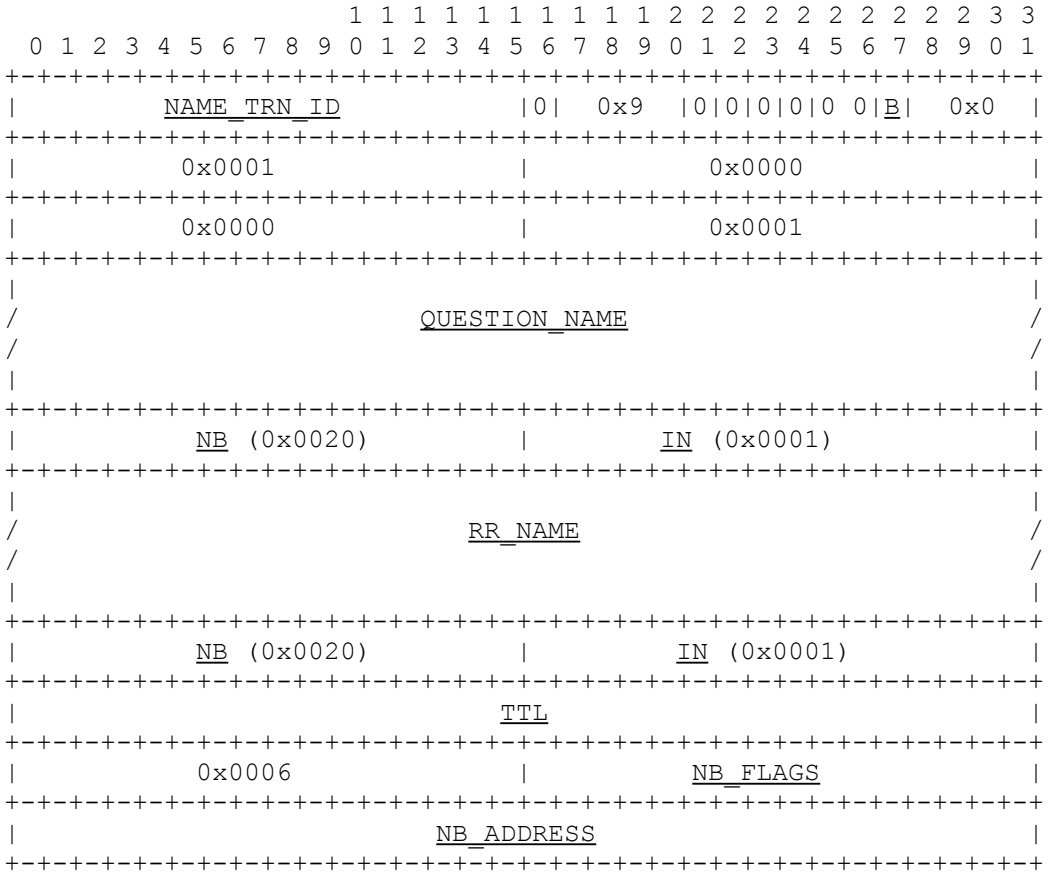


Since the RR_NAME is the same name as the QUESTION_NAME, the RR_NAME representation must use pointers to the QUESTION_NAME name's labels to guarantee the length of the datagram is less than the maximum 576 bytes. See section above on name formats and also page 31 and 32 of RFC 883, Domain Names - Implementation and Specification, for a complete description of compressed name label pointers.

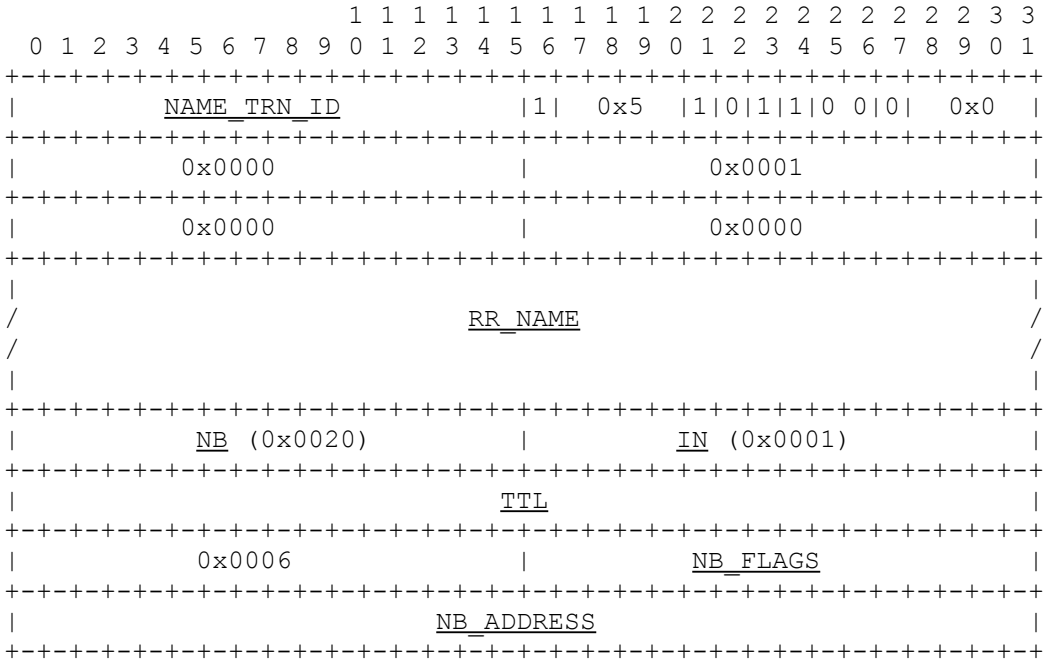
Name Overwrite Request & Demand



Name Refresh Request

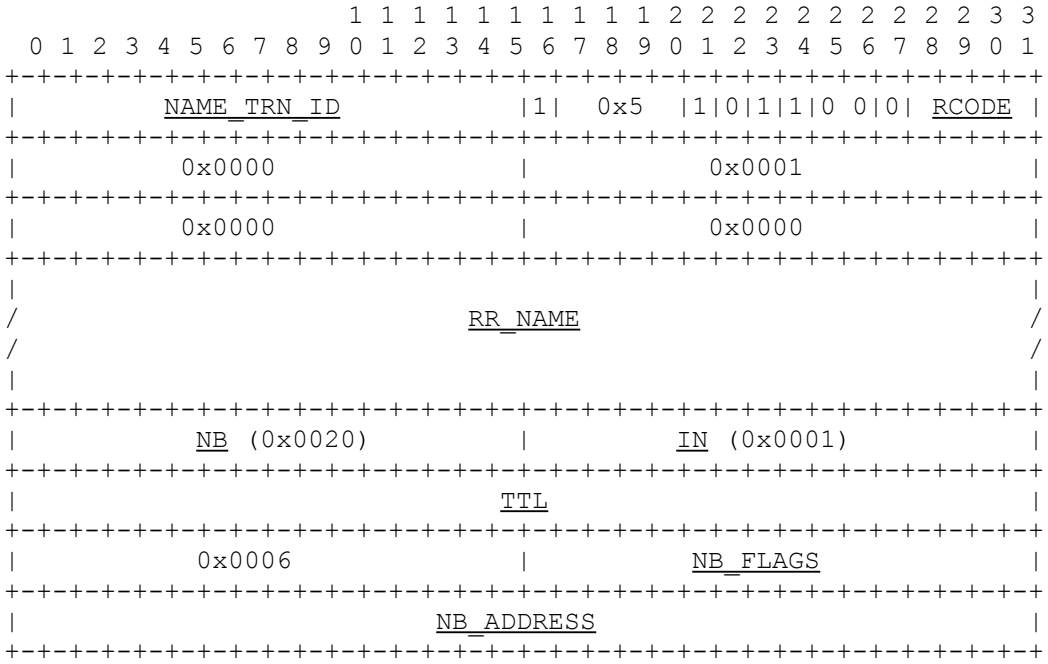


Positive Name Registration Response



RFC-1002 Protocol Standard for a NetBIOS Service - Name Service Packets

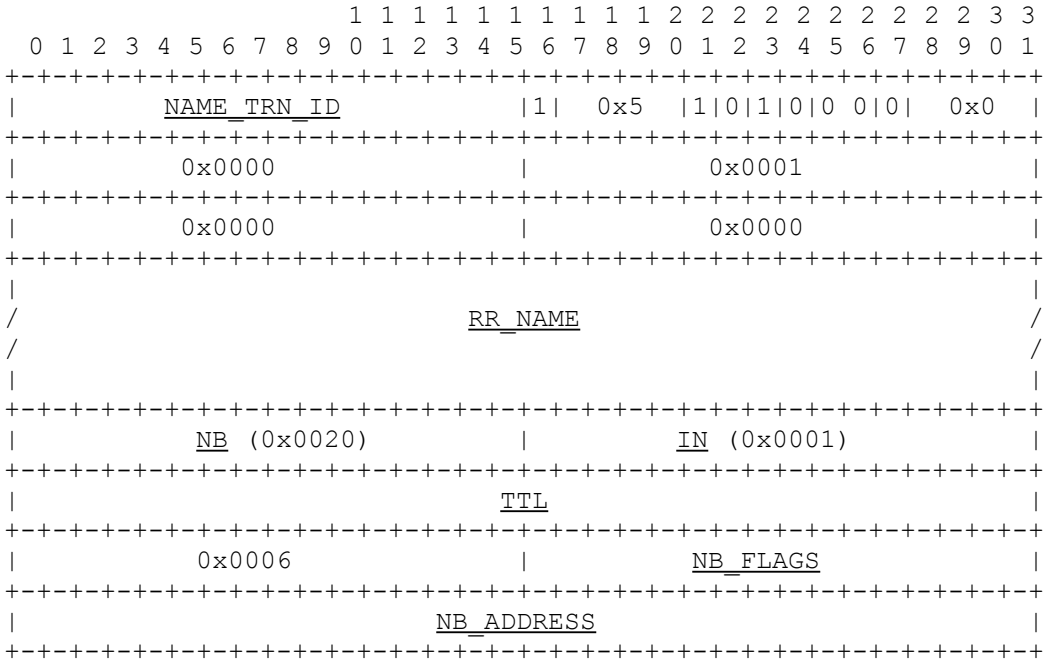
Negative Name Registration Response



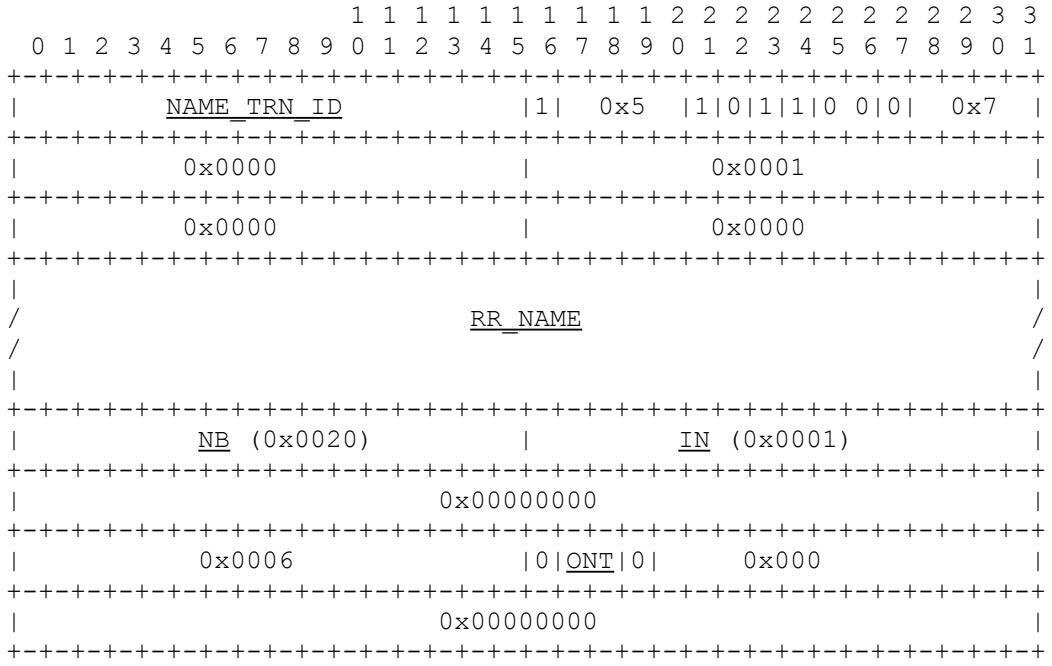
RCODE field values:

Symbol	Value	Description:
FMT_ERR	0x1	Format Error. Request was invalidly formatted.
SRV_ERR	0x2	Server failure. Problem with NBNS, cannot process name.
IMP_ERR	0x4	Unsupported request error. Allowable only for challenging NBNS when gets an Update type registration request.
RFS_ERR	0x5	Refused error. For policy reasons server will not register this name from this host.
ACT_ERR	0x6	Active error. Name is owned by another node.
CFT_ERR	0x7	Name in conflict error. A UNIQUE name is owned by more than one node.

End-Node Challenge Registration Response

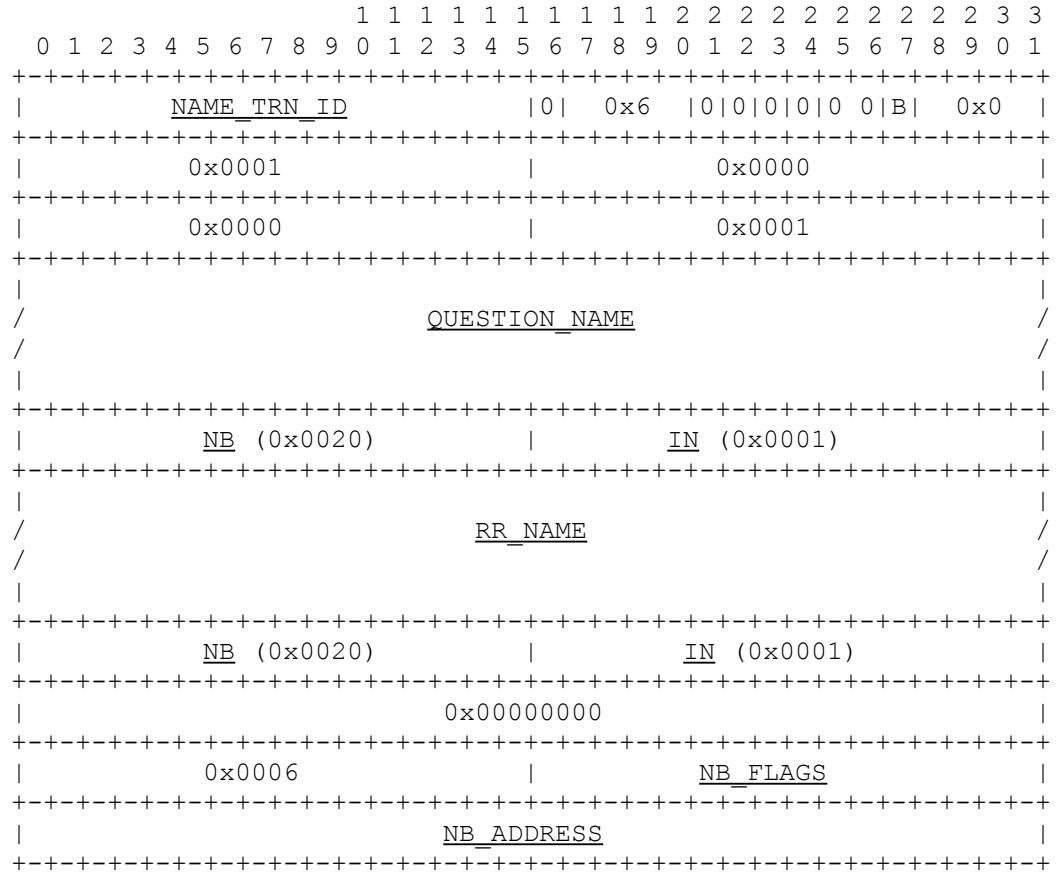


Name Conflict Demand



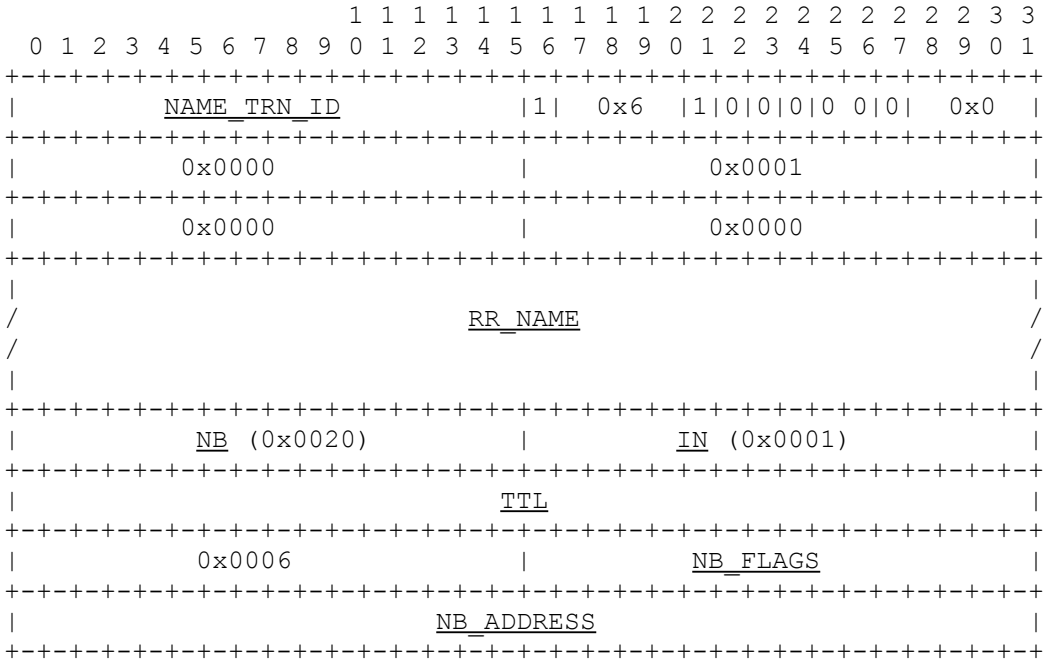
This packet is identical to a NEGATIVE NAME REGISTRATION RESPONSE with RCODE = CFT_ERR.

Name Release Request & Demand

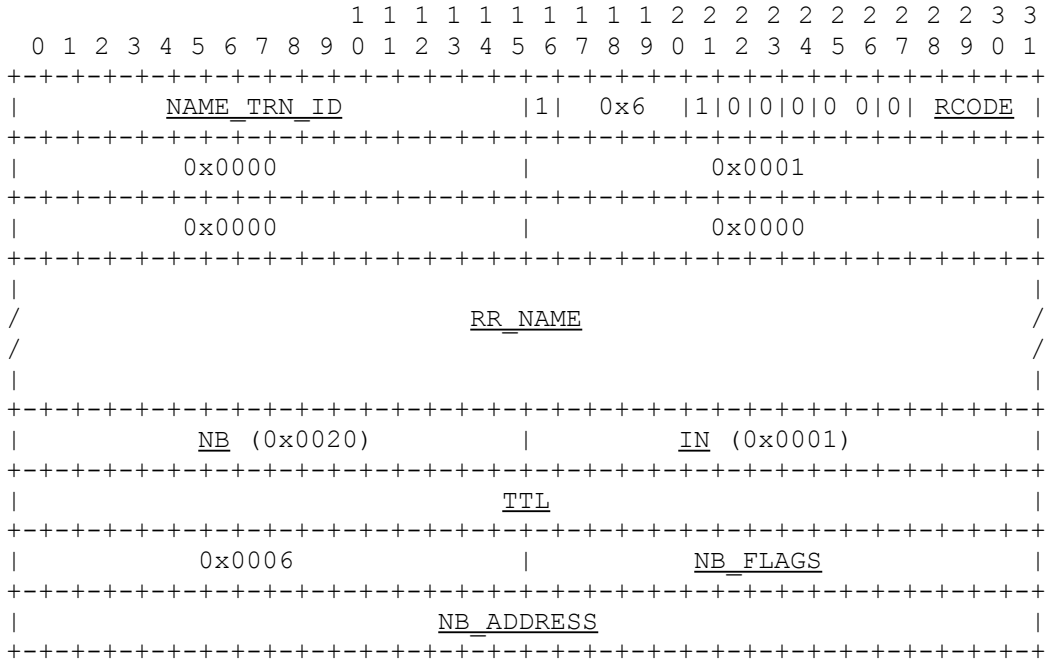


Since the RR_NAME is the same name as the QUESTION_NAME, the RR_NAME representation must use label string pointers to the QUESTION_NAME labels to guarantee the length of the datagram is less than the maximum 576 bytes. This is the same condition as with the NAME REGISTRATION REQUEST.

Positive Name Release Response



Negative Name Release Response

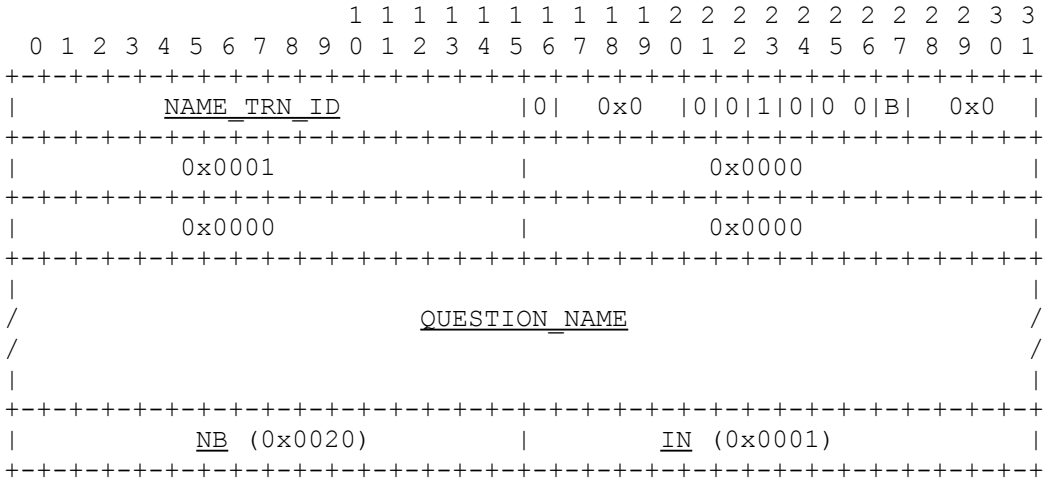


RCODE field values:

Symbol	Value	Description:
FMT_ERR	0x1	Format Error. Request was invalidly formatted.
SRV_ERR	0x2	Server failure. Problem with NBNS, cannot process name.
RFS_ERR	0x5	Refused error. For policy reasons server will not release this name from this host.
ACT_ERR	0x6	Active error. Name is owned by another node. Only that node may release it. A NetBIOS Name Server can optionally allow a node to release a name it does not own. This would facilitate detection of inactive names for nodes that went down silently.

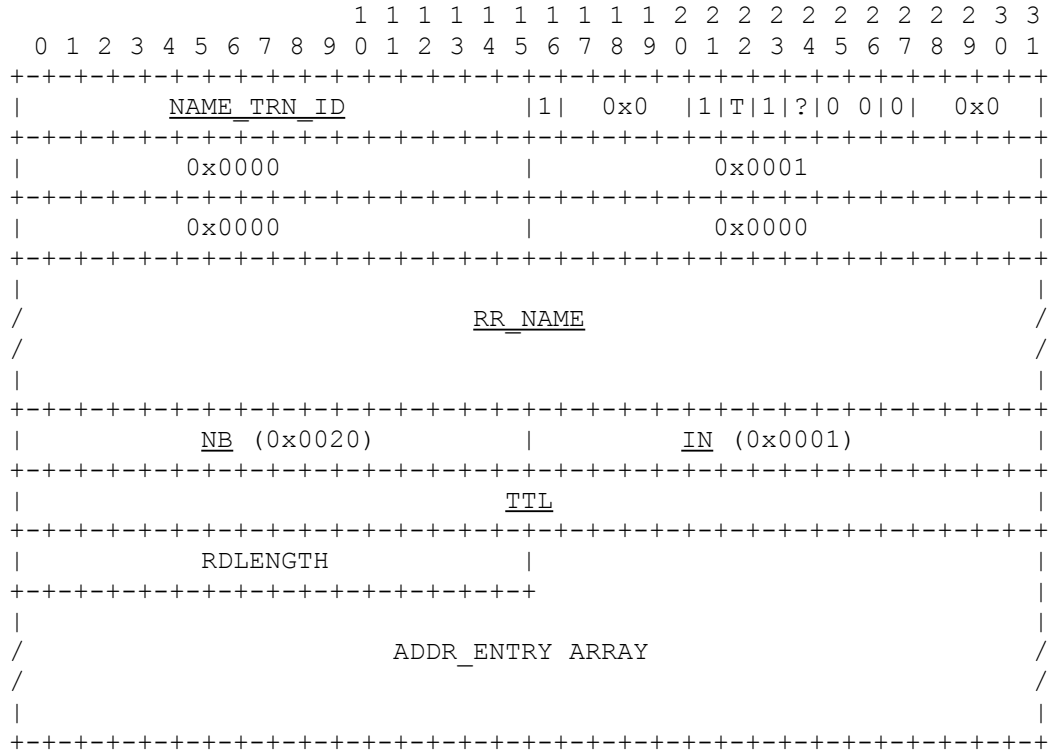
RFC-1002 Protocol Standard for a NetBIOS Service - Name Service Packets

Name Query Request



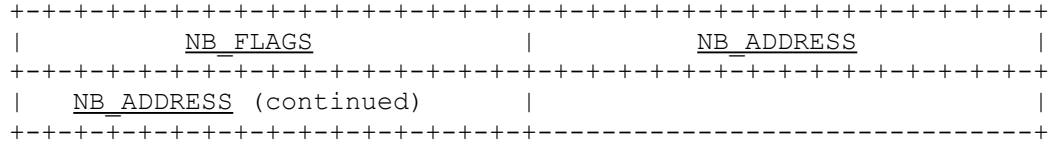
RFC-1002 Protocol Standard for a NetBIOS Service - Name Service Packets

Positive Name Query Response



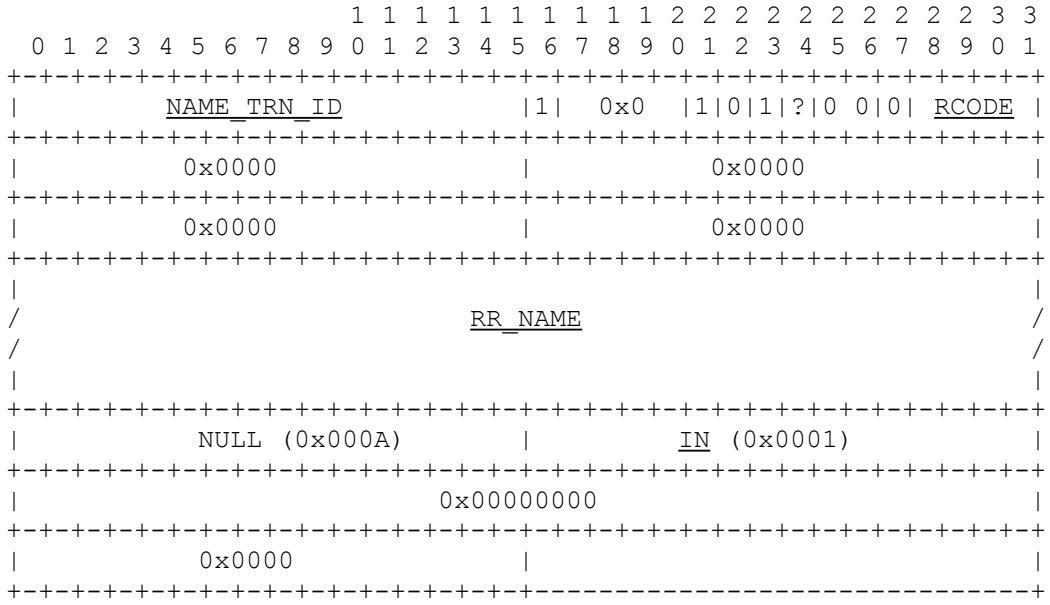
The ADDR_ENTRY ARRAY a sequence of zero or more ADDR_ENTRY records. Each ADDR_ENTRY record represents an owner of a name. For group names there may be multiple entries. However, the list may be incomplete due to packet size limitations. Bit 22, "T", will be set to indicate truncated data.

Each ADDR_ENTRY has the following format:



RFC-1002 Protocol Standard for a NetBIOS Service - Name Service Packets

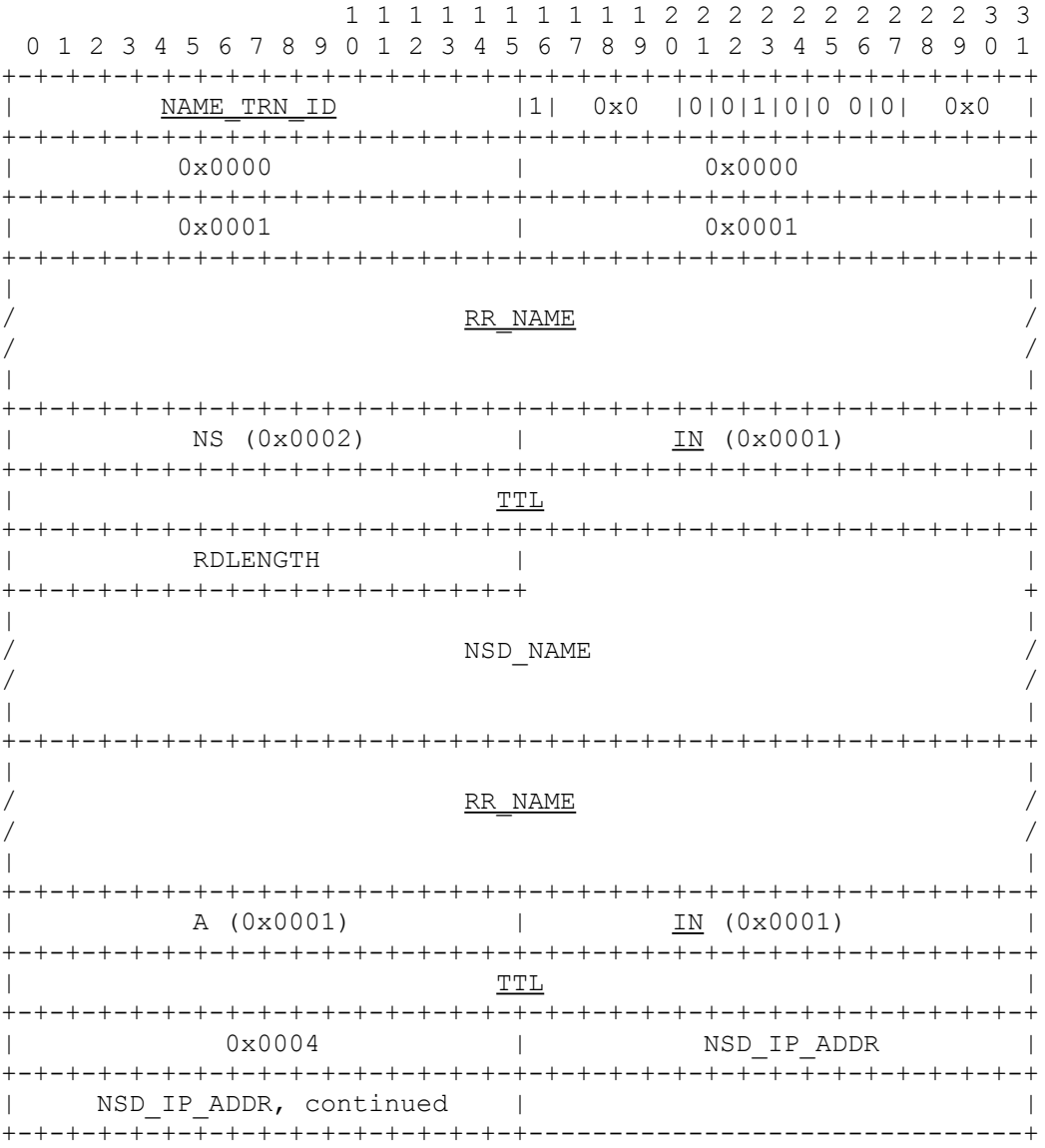
Negative Name Query Response



RCODE field values:

Symbol	Value	Description
FMT_ERR	0x1	Format Error. Request was invalidly formatted.
SRV_ERR	0x2	Server failure. Problem with NBNS, cannot process name.
NAM_ERR	0x3	Name Error. The name requested does not exist.
IMP_ERR	0x4	Unsupported request error. Allowable only for challenging NBNS when gets an Update type registration request.
RFS_ERR	0x5	Refused error. For policy reasons server will not register this name from this host.

Redirect Name Query Response



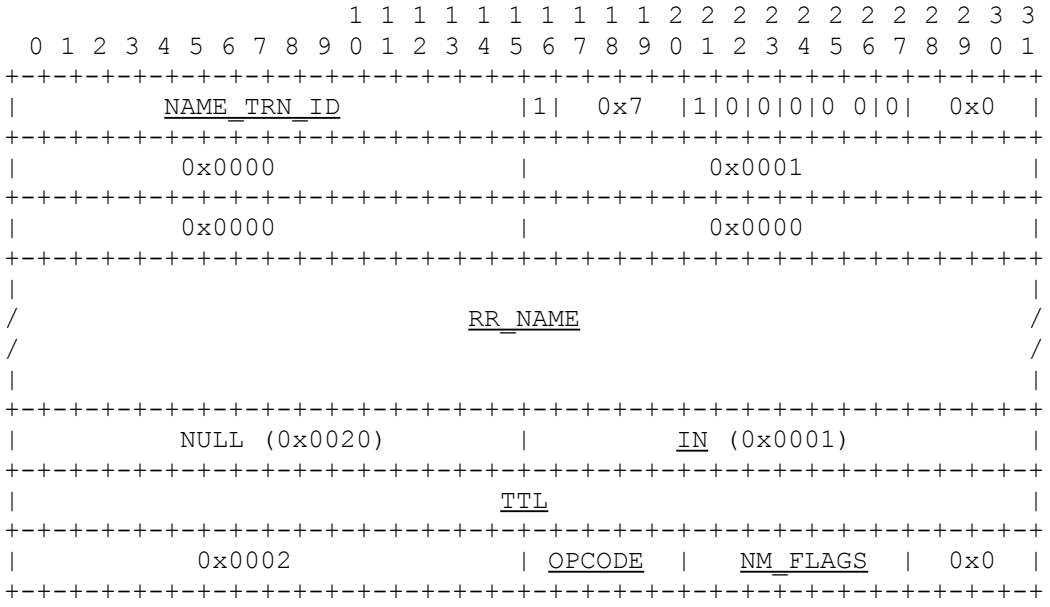
An end node responding to a NAME QUERY REQUEST always responds with the AA and RA bits set for both the NEGATIVE and POSITIVE NAME QUERY RESPONSE packets. An end node never sends a REDIRECT NAME QUERY RESPONSE packet.

When the requestor receives the REDIRECT NAME QUERY RESPONSE it must reiterate the NAME QUERY REQUEST to the NBNS specified by the NSD_IP_ADDR field of the A type RESOURCE RECORD in the ADDITIONAL section of the response packet. This is an optional packet for the NBNS.

The NSD_NAME and the RR_NAME in the ADDITIONAL section of the response packet are the same name. Space can be optimized if label string pointers are used in the RR_NAME which point to the labels in the NSD_NAME.

The RR_NAME in the AUTHORITY section is the name of the domain the NBNS called by NSD_NAME has authority over.

Wait for Acknowledgement (Wack) Response

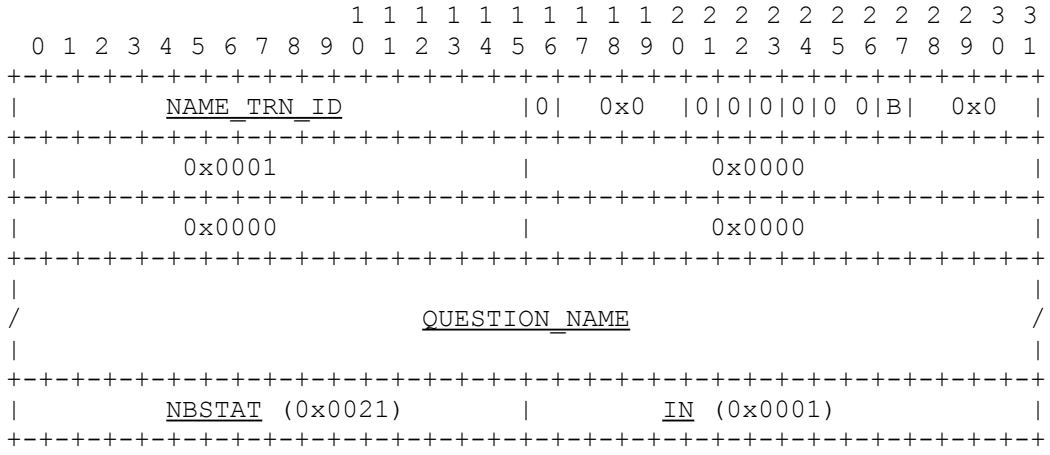


The NAME_TRN_ID of the WACK RESPONSE packet is the same NAME_TRN_ID of the request that the NBNS is telling the requestor to wait longer to complete. The RR_NAME is the name from the request, if any. If no name is available from the request then it is a null name, single byte of zero.

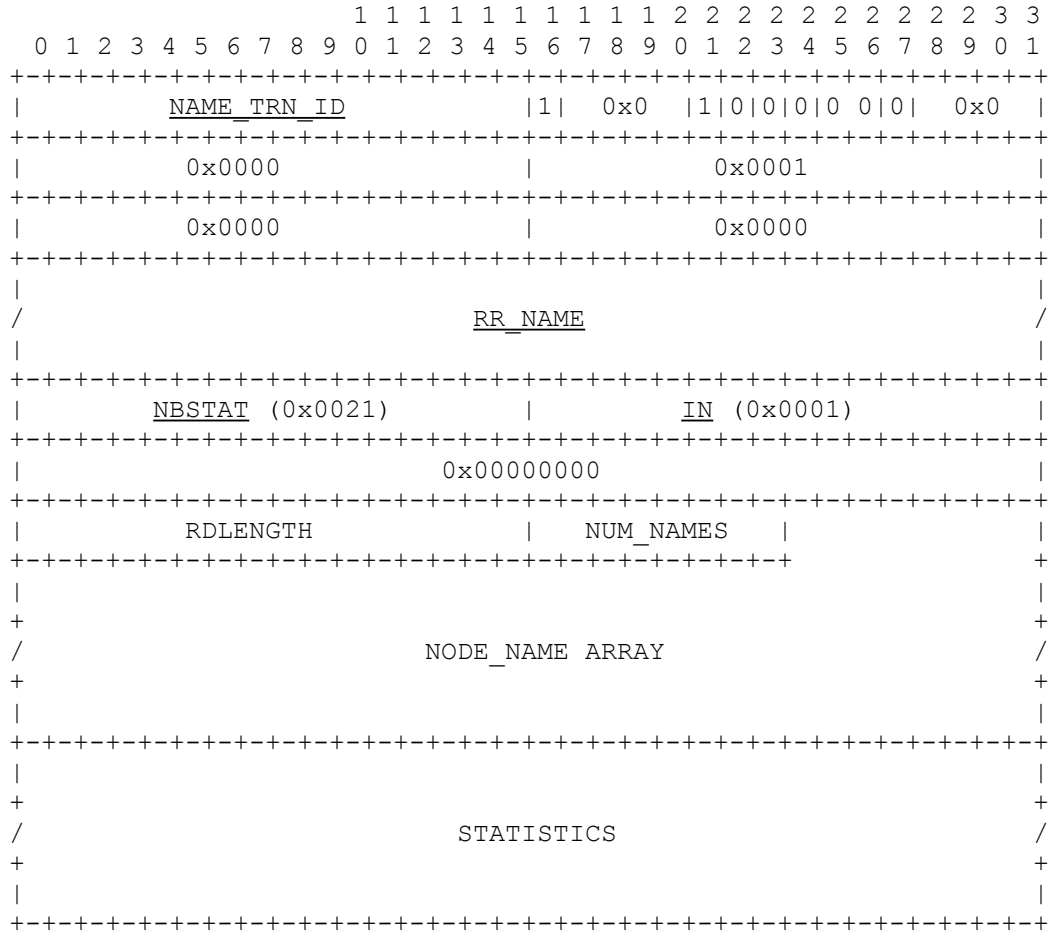
The TTL field of the ResourceRecord is the new time to wait, in seconds, for the request to complete. The RDATA field contains the OPCODE and NM_FLAGS of the request.

A TTL value of 0 means that the NBNS can not estimate the time it may take to complete a response.

Node Status Request

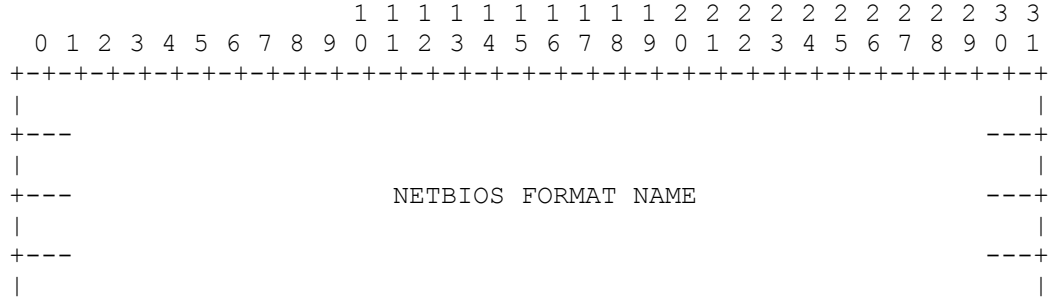


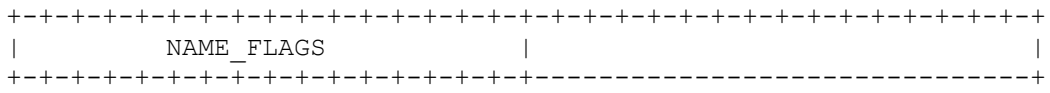
Node Status Response



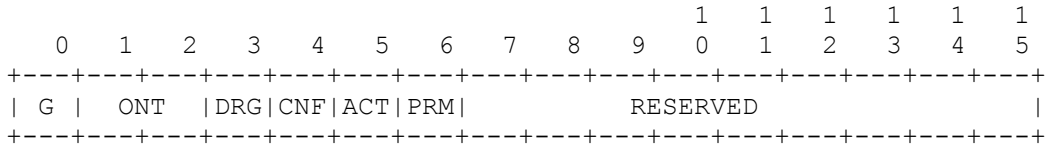
The NODE_NAME ARRAY is an array of zero or more NUM_NAMES entries of NODE_NAME records. Each NODE_NAME entry represents an active name in the same NetBIOS scope as the requesting name in the local name table of the responder. RR_NAME is the requesting name.

NODE_NAME Entry:





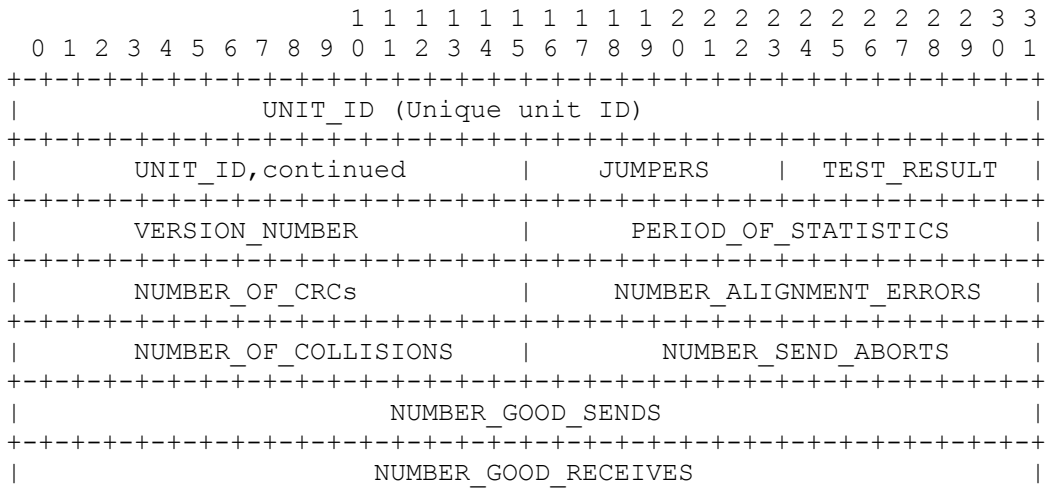
The NAME_FLAGS field:

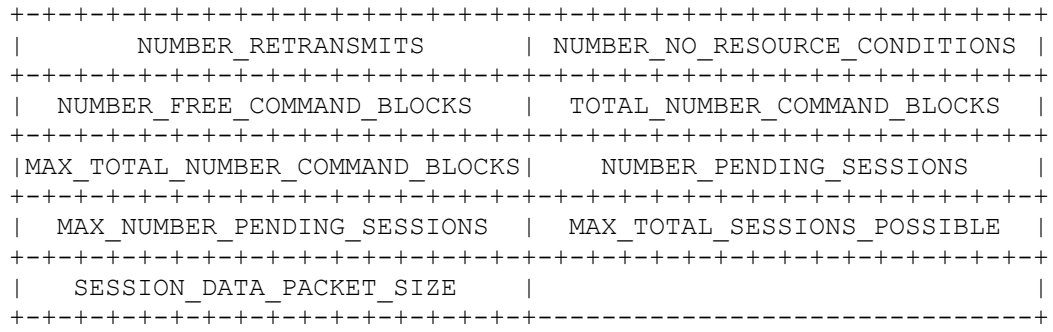


The NAME_FLAGS field is defined as:

Symbol	Bit(s)	Description:
RESERVED	7-15	Reserved for future use. Must be zero (0).
PRM	6	Permanent Name Flag. If one (1) then entry is for the permanent node name. Flag is zero (0) for all other names.
ACT	5	Active Name Flag. All entries have this flag set to one (1).
CNF	4	Conflict Flag. If one (1) then name on this node is in conflict.
DRG	3	Deregister Flag. If one (1) then this name is in the process of being deleted.
ONT	1,2	Owner Node Type: 00 = B node 01 = P node 10 = M node 11 = Reserved for future use
G	0	Group Name Flag. If one (1) then the name is a GROUP NetBIOS name. If zero (0) then it is a UNIQUE NetBIOS name.

STATISTICS Field of the NODE STATUS RESPONSE:




NUMBER_RETRANSMITS
NUMBER_NO_RESOURCE_CONDITIONS
NUMBER_FREE_COMMAND_BLOCKS
TOTAL_NUMBER_COMMAND_BLOCKS
MAX_TOTAL_NUMBER_COMMAND_BLOCKS
NUMBER_PENDING_SESSIONS
MAX_NUMBER_PENDING_SESSIONS
MAX_TOTAL_SESSIONS_POSSIBLE
SESSION_DATA_PACKET_SIZE

RFC-1002 Protocol Standard for a NetBIOS Service - Packet Descriptions

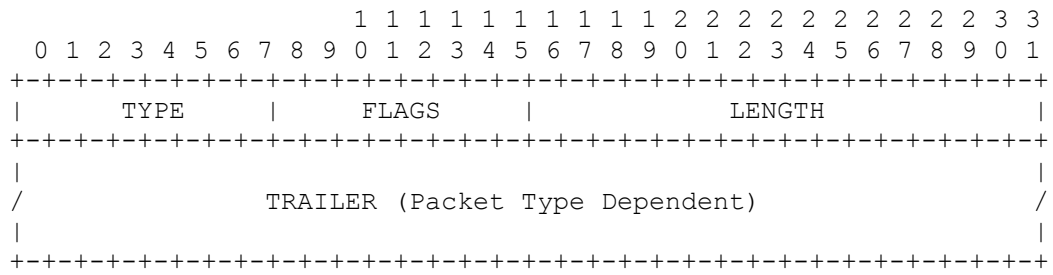
Session Service Packets

RFC-1002 Protocol Standard for a NetBIOS Service - Session Service Packets

General Format of Session Packets

All session service messages are sent over a TCP connection.

All session packets are of the following general structure:



The TYPE, FLAGS, and LENGTH fields are present in every session packet.

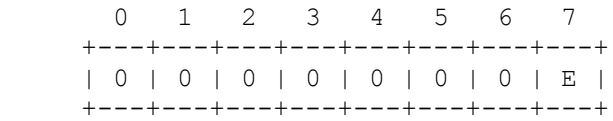
The LENGTH field is the number of bytes following the LENGTH field. In other words, LENGTH is the combined size of the TRAILER field(s). For example, the POSITIVE SESSION RESPONSE packet always has a LENGTH field value of zero (0000) while the RETARGET SESSION RESPONSE always has a LENGTH field value of six (0006).

One of the bits of the FLAGS field acts as an additional, high- order bit for the LENGTH field. Thus the cumulative size of the trailer field(s) may range from 0 to 128K bytes.

Session Packet Types (in hexidecimal):

- 00 - SESSION MESSAGE
- 81 - SESSION REQUEST
- 82 - POSITIVE SESSION RESPONSE
- 83 - NEGATIVE SESSION RESPONSE
- 84 - RETARGET SESSION RESPONSE
- 85 - SESSION KEEP ALIVE

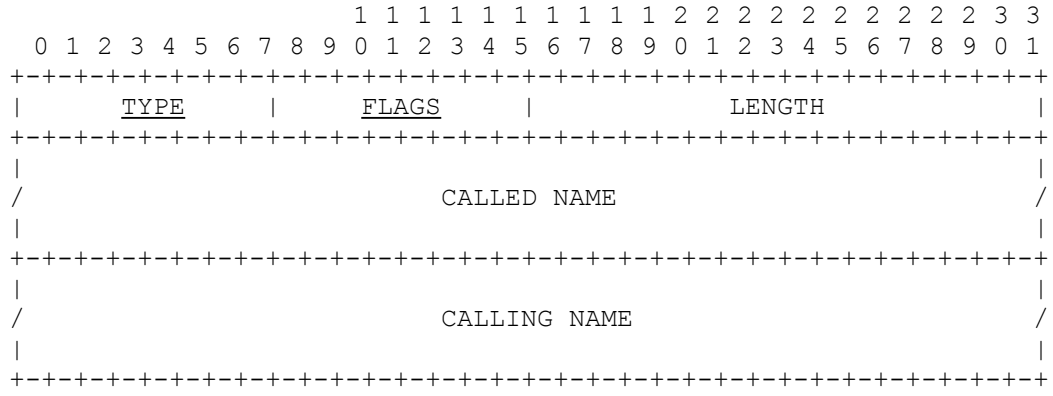
Bit definitions of the FLAGS field:



Symbol	Bit(s)	Description
E	7	Length extension, used as an additional, high-order bit on the LENGTH field.
RESERVED	0-6	Reserved, must be zero (0)

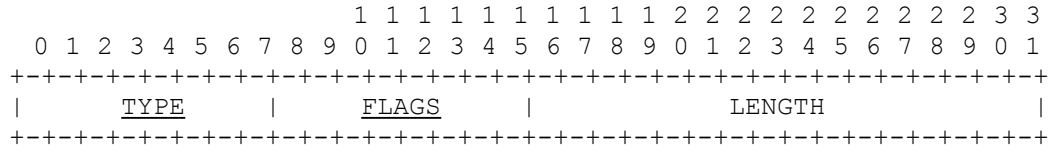
RFC-1002 Protocol Standard for a NetBIOS Service - Session Service Packets

Session Request Packet



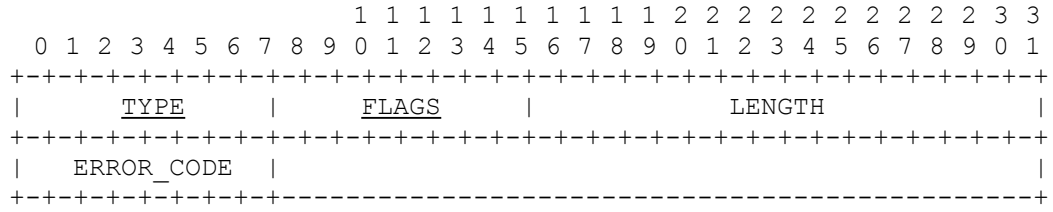
RFC-1002 Protocol Standard for a NetBIOS Service - Session Service Packets

Positive Session Response Packet



RFC-1002 Protocol Standard for a NetBIOS Service - Session Service Packets

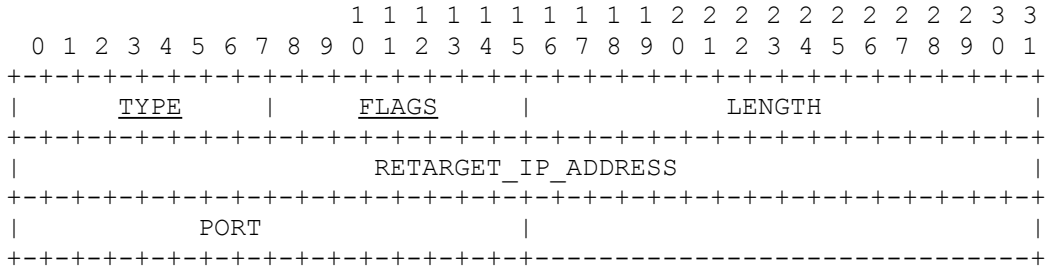
Negative Session Response Packet



NEGATIVE SESSION RESPONSE packet error code values (in hexidecimal):

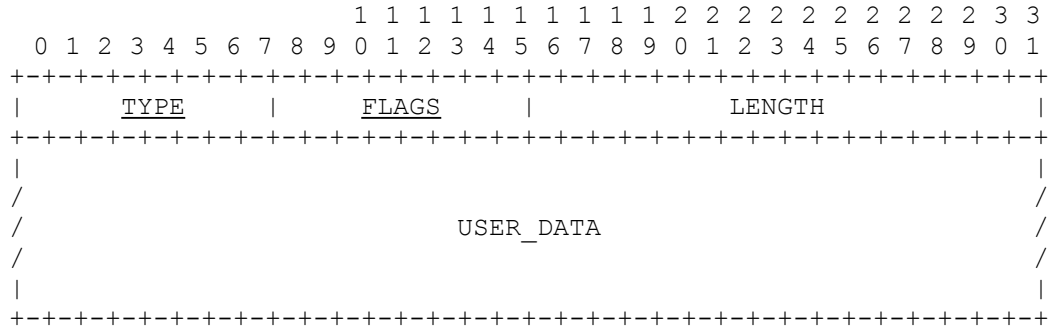
80 - Not listening on called name
81 - Not listening for calling name
82 - Called name not present
83 - Called name present, but insufficient resources
8F - Unspecified error

Session Retarget Response Packet



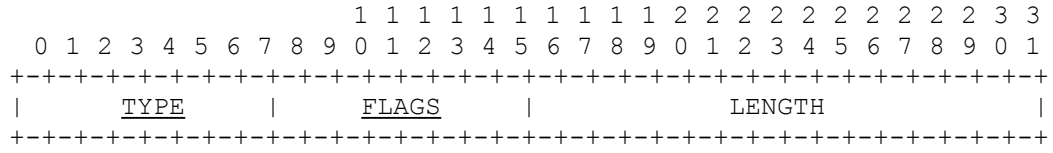
RFC-1002 Protocol Standard for a NetBIOS Service - Session Service Packets

Session Message Packet



RFC-1002 Protocol Standard for a NetBIOS Service - Session Service Packets

Session Keep Alive Packet

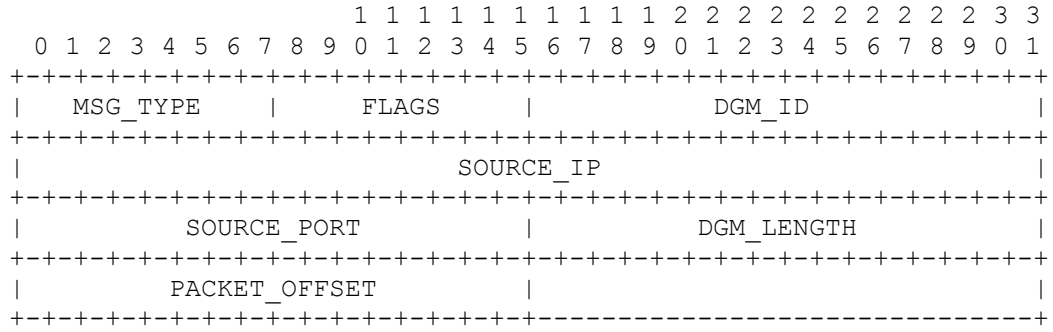


RFC-1002 Protocol Standard for a NetBIOS Service - Packet Descriptions

Datagram Service Packets

RFC-1002 Protocol Standard for a NetBIOS Service - Datagram Service Packets

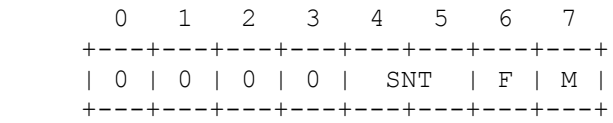
NetBIOS Datagram Header



MSG_TYPE values (in hexidecimal):

10 - DIRECT_UNIQUE DATAGRAM
11 - DIRECT_GROUP DATAGRAM
12 - BROADCAST DATAGRAM
13 - DATAGRAM ERROR
14 - DATAGRAM QUERY REQUEST
15 - DATAGRAM POSITIVE QUERY RESPONSE
16 - DATAGRAM NEGATIVE QUERY RESPONSE

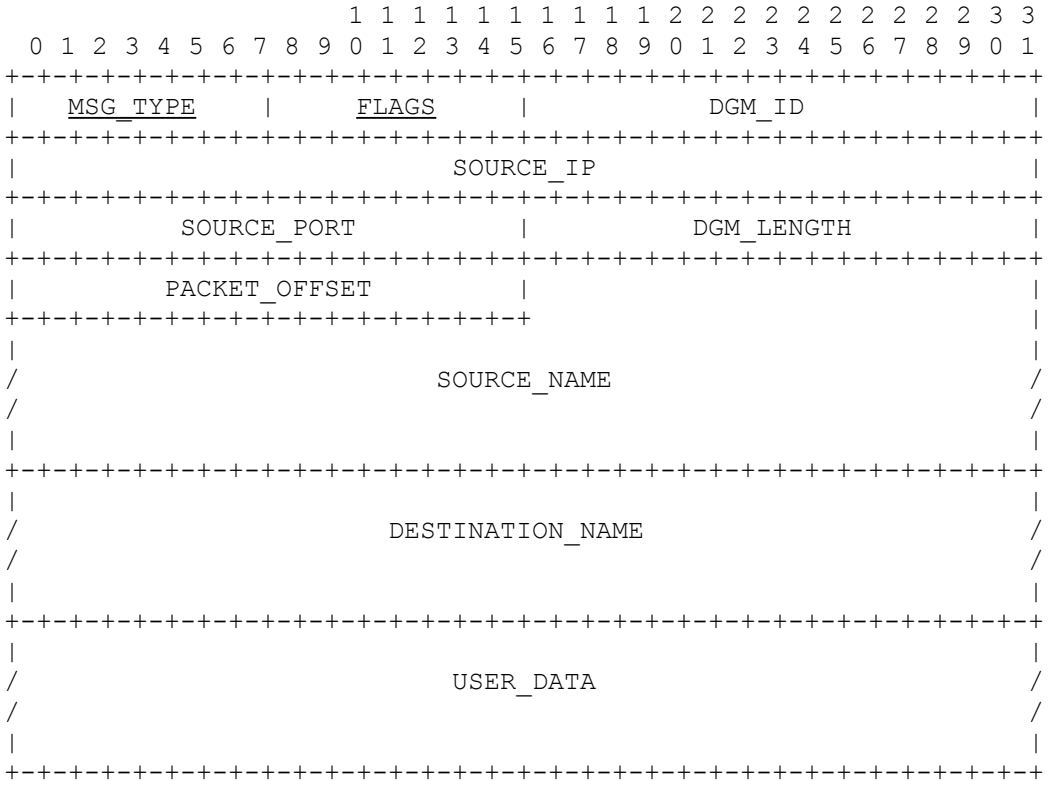
Bit definitions of the FLAGS field:



Symbol	Bit(s)	Description
M	7	MORE flag, If set then more NetBIOS datagram fragments follow.
F	6	FIRST packet flag, If set then this is first (and possibly only) fragment of NetBIOS datagram
SNT	4,5	Source End-Node type: 00 = B node 01 = P node 10 = M node 11 = NBDD
RESERVED	0-3	Reserved, must be zero (0)

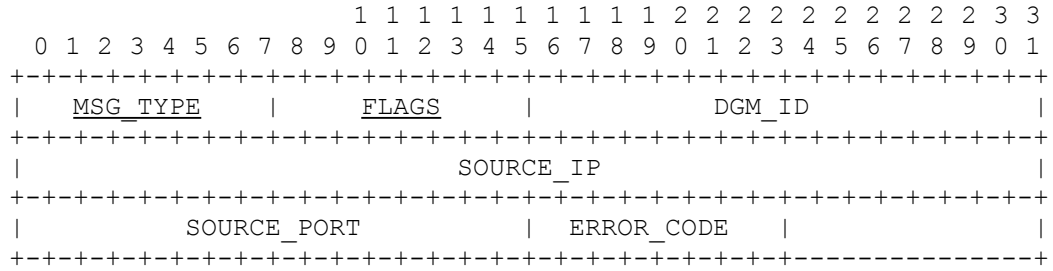
RFC-1002 Protocol Standard for a NetBIOS Service - Session Service Packets

Direct_Unique, Direct_Group, & Broadcast Datagram



RFC-1002 Protocol Standard for a NetBIOS Service - Session Service Packets

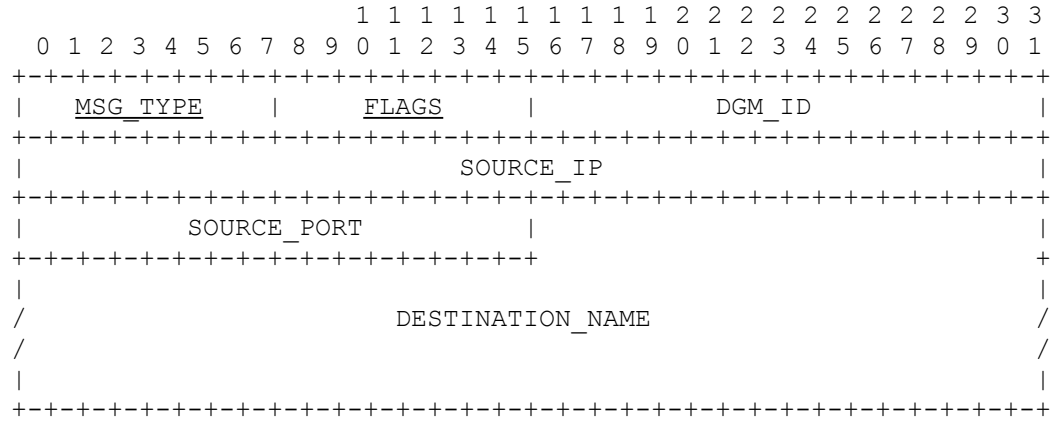
Datagram Error Packet



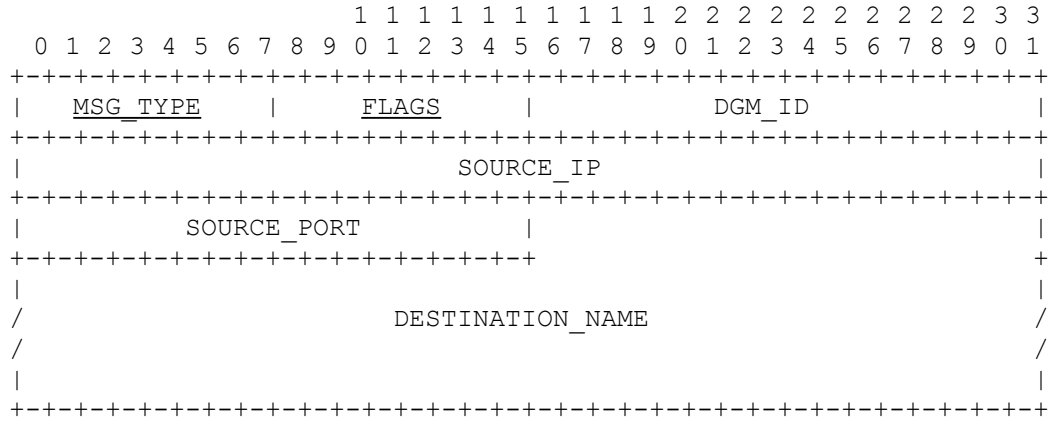
ERROR_CODE values (in hexidecimal):

82 - DESTINATION NAME NOT PRESENT
83 - INVALID SOURCE NAME FORMAT
84 - INVALID DESTINATION NAME FORMAT

4.4.4. Datagram Query Request



Datagram Positive and Negative Query Response



RFC-1002 Protocol Standard for a NetBIOS Service - Detailed Specifications

Protocol Descriptions

Name Service Protocols
Session Service Protocolsh1002_SessionProtocols
NetBIOS Datagram Service Protocols

Name Service Protocols

A REQUEST packet is always sent to the well known UDP port - NAME_SERVICE_UDP_PORT. The destination address is normally either the IP broadcast address or the address of the NBNS - the address of the NBNS server it set up at initialization time. In rare cases, a request packet will be sent to an end node, e.g. a NAME QUERY REQUEST sent to "challenge" a node.

A RESPONSE packet is always sent to the source UDP port and source IP address of the request packet.

A DEMAND packet must always be sent to the well known UDP port - NAME_SERVICE_UDP_PORT. There is no restriction on the target IP address.

Terms used in this section:

tid Transaction ID. This is a value composed from the requestor's IP address and a unique 16 bit value generated by the originator of the transaction.

B-Node Activity

RFC-1002 Protocol Standard for a NetBIOS Service - B-Node Activity

B-Node Add Name

```
PROCEDURE add_name(newname)
   /*
    * Host initiated processing for a B node
    */
BEGIN
        REPEAT
            /* build name service packet */

            ONT = B_NODE; /* broadcast node */
            G = UNIQUE;    /* unique name */
            TTL = 0;

            broadcast NAME REGISTRATION REQUEST packet;

            /*
             * remote node(s) will send response packet
             * if applicable
             */

            pause(BCAST_REQ_RETRY_TIMEOUT);

        UNTIL response packet is received or retransmit count has been exceeded
        IF no response packet was received THEN
        BEGIN /* no response */
            /*
             * build packet
             */

            ONT = B_NODE; /* broadcast node */
            G = UNIQUE;    /* unique name */
            TTL = 0;
            /*
             * Let other nodes known you have the name
             */

            broadcast NAME UPDATE REQUEST packet;
            /* name can be added to local name table */
            return success;
        END /* no response */
        ELSE
        BEGIN /* got response */

            /*
             * Match return transaction id
             * against tid sent in request
             */

            IF NOT response tid = request tid THEN
            BEGIN
            ignore response packet;
```

```
            END
            ELSE
            CASE packet type OF

            NEGATIVE NAME REGISTRATION RESPONSE:

                return failure; /* name cannot be added */

            POSITIVE NAME REGISTRATION RESPONSE:
            END-NODE CHALLENGE NAME REGISTRATION RESPONSE:

                /*
                * B nodes should normally not get this
                * response.
                */

                ignore packet;
            END /* case */;
        END /* got response */
END /* procedure */
```

B-Node Add_Group Name

```
PROCEDURE add_group_name(newname)

    /*
     * Host initiated processing for a B node
     */

BEGIN
          /*
           * same as for a unique name with the
           * exception that the group bit (G) must
           * be set in the request packets.
           */
          ...
          G = GROUP;
          ...
          ...

          /*
           * broadcast request ...
           */
END
```

RFC-1002 Protocol Standard for a NetBIOS Service - B-Node Activity

B-Node Find_Name

```
PROCEDURE find_name(name)

    /*
     * Host initiated processing for a B node
     */

BEGIN

        REPEAT
            /*
             * build packet
             */
            ONT = B;
            TTL = 0;
            G = DONT CARE;

            broadcast NAME QUERY REQUEST packet;

            /*
             * a node might send response packet
             */

            pause(BCAST_REQ_RETRY_TIMEOUT);
        UNTIL response packet received OR
            max transmit threshold exceeded

        IF no response packet received THEN
            return failure;
        ELSE
        IF NOT response tid = request tid THEN
            ignore packet;
        ELSE
        CASE packet type OF
        POSITIVE NAME QUERY RESPONSE:
            /*
             * Start a timer to detect conflict.
             *
             * Be prepared to detect conflict if
             * any more response packets are received.
             *
             */

            save response as authoritative response;
            start_timer(CONFLICT_TIMER);
            return success;

        NEGATIVE NAME QUERY RESPONSE:
        REDIRECT NAME QUERY RESPONSE:

            /*
```

```
                * B Node should normally not get either
                * response.
                */

                ignore response packet;

        END /* case */
END /* procedure */
```

B Node Name Release

```
PROCEDURE delete_name (name)
BEGIN

        REPEAT

            /*
             * build packet
             */
            ...

            /*
             * send request
             */

            broadcast NAME RELEASE REQUEST packet;

            /*
             * no response packet expected
             */

            pause(BCAST_REQ_RETRY_TIMEOUT);

        UNTIL retransmit count has been exceeded
END /* procedure */
```

B-Node Incoming Packet Processing

Following processing is done when broadcast or unicast packets are received at the NAME_SERVICE_UDP_PORT.

```
PROCEDURE process_incoming_packet(packet)
    /*
     * Processing initiated by incoming packets for a B node
     */

BEGIN
          /*
           * Note: response packets are always sent
           * to:
           * source IP address of request packet
           * source UDP port of request packet
           */

          CASE packet type OF

          NAME REGISTRATION REQUEST (UNIQUE):
               IF name exists in local name table THEN
                    send NEGATIVE NAME REGISTRATION RESPONSE ;
          NAME REGISTRATION REQUEST (GROUP):
               IF name exists in local name table THEN
               BEGIN
                    IF local entry is a unique name THEN
                         send NEGATIVE NAME REGISTRATION RESPONSE ;
               END
          NAME QUERY REQUEST:
               IF name exists in local name table THEN
               BEGIN
                    build response packet;
                    send POSITIVE NAME QUERY RESPONSE;
          POSITIVE NAME QUERY RESPONSE:
               IF name conflict timer is not active THEN
               BEGIN
                    /*
                     * timer has expired already...   ignore this
                     * packet
                     */

                    return;
               END
               ELSE /* timer is active */
               IF a response for this name has previously been
                    received THEN
               BEGIN /* existing entry */

                    /*
                     * we sent out a request packet, and
                     * have already received (at least)
```

```
            * one response
            *
            * Check if conflict exists.
            * If so, send out a conflict packet.
            *
            * Note: detecting conflict does NOT
            * affect any existing sessions.
            *
            */

           /*
            * Check for name conflict.
            * See "Name Conflict" in Concepts and Methods
            */
           check saved authoritative response against
                 information in this response packet;
           IF conflict detected THEN
           BEGIN
                 unicast NAME CONFLICT DEMAND packet;
                 IF entry exists in cache THEN
                 BEGIN
                       remove entry from cache;
                 END
           END
      END /* existing entry */
      ELSE
      BEGIN
           /*
            * Note: If this was the first response
            * to a name query, it would have been
            * handled in the
            * find_name() procedure.
            */

           ignore packet;
      END
NAME CONFLICT DEMAND:
      IF name exists in local name table THEN
      BEGIN
           mark name as conflict detected;

           /*
            * a name in the state "conflict detected"
            * does not "logically" exist on that node.
            * No further session will be accepted on
            * that name.
            * No datagrams can be sent against that name.
            * Such an entry will not be used for
            * purposes of processing incoming request
            * packets.
            * The only valid user NetBIOS operation
            * against such a name is DELETE NAME.
            */
      END
NAME RELEASE REQUEST:
```

```
            IF caching is being done THEN
            BEGIN
                remove entry from cache;
            END
        NAME UPDATE REQUEST:
            IF caching is being done THEN
            BEGIN
                IF entry exists in cache already,
                        update cache;
                ELSE IF name is "interesting" THEN
                BEGIN
                        add entry to cache;
                END
            END

        NODE STATUS REQUEST:
            IF name exists in local name table THEN
            BEGIN
                /*
                 * send only those names that are
                 * in the same scope as the scope
                 * field in the request packet
                 */

                send NODE STATUS RESPONSE;
            END
END
```

P-Node Activity

All packets sent or received by P nodes are unicast UDP packets. A P node sends name service requests to the NBNS node that is specified in the P-node configuration.

RFC-1002 Protocol Standard for a NetBIOS Service - P-Node Activity

P-Node Add_Name

```
PROCEDURE add_name(newname)

    /*
     * Host initiated processing for a P node
     */

BEGIN

          REPEAT
               /*
                * build packet
                */

               ONT = P;
               G = UNIQUE;
               ...

               /*
                * send request
                */

               unicast NAME REGISTRATION REQUEST packet;

               /*
                * NBNS will send response packet
                */

               IF receive a WACK RESPONSE THEN
                    pause(time from TTL field of response);
               ELSE
                    pause(UCAST_REQ_RETRY_TIMEOUT);
          UNTIL response packet is received OR
               retransmit count has been exceeded

          IF no response packet was received THEN
          BEGIN /* no response */
               /*
                * NBNS is down.   Cannot claim name.
                */

               return failure; /* name cannot be claimed */
          END /* no response */
          ELSE
          BEGIN /* response */
               IF NOT response tid = request tid THEN
               BEGIN
               /*   Packet may belong to another transaction   */
               ignore response packet;
               END
               ELSE
```

```
CASE packet type OF

POSITIVE NAME REGISTRATION RESPONSE:

     /*
     * name can be added
     */

     adjust refresh timeout value, TTL, for this name;
     return success;         /* name can be added */

NEGATIVE NAME REGISTRATION RESPONSE:
     return failure; /* name cannot be added */

END-NODE CHALLENGE REGISTRATION REQUEST:
BEGIN /* end node challenge */

     /*
     * The response packet has in it the
     * address of the presumed owner of the
     * name.   Challenge that owner.
     * If owner either does not
     * respond or indicates that he no longer
     * owns the name, claim the name.
     * Otherwise, the name cannot be claimed.
     *
     */

     REPEAT
     /*
      * build packet
      */
     ...

     unicast NAME QUERY REQUEST packet to the
            address contained in the END NODE
            CHALLENGE RESPONSE packet;

     /*
      * remote node may send response packet
      */

     pause(UCAST_REQ_RETRY_TIMEOUT);

     UNTIL response packet is received or
         retransmit count has been exceeded
     IF no response packet is received OR
            NEGATIVE NAME QUERY RESPONSE packet
            received THEN
     BEGIN /* update */

     /*
      * name can be claimed
      */
```

```
                REPEAT

                    /*
                     * build packet
                     */
                     ...

                    unicast NAME UPDATE REQUEST to NBNS;

                    /*
                     * NBNS node will send response packet
                     */

                    IF receive a WACK RESPONSE THEN
                            pause(time from TTL field of response);
                    ELSE
                            pause(UCAST_REQ_RETRY_TIMEOUT);
                UNTIL response packet is received or
                    retransmit count has been exceeded
                IF no response packet received THEN
                BEGIN /* no response */

                     /*
                      * name could not be claimed
                      */

                     return failure;
                END /* no response */
                ELSE
                CASE packet type OF
                     POSITIVE NAME REGISTRATION RESPONSE:
                            /*
                             * add name
                             */
                            return success;
                     NEGATIVE NAME REGISTRATION RESPONSE:

                            /*
                             * you lose   ...
                             */
                            return failure;
                     END /* case */
                END /* update */
                ELSE

                /*
                * received a positive response to the "challenge"
                * Remote node still has name
                */

                return failure;
            END /* end node challenge */
        END /* response */
END /* procedure */
```

P-Node Add Group Name

```
PROCEDURE add_group_name(newname)

    /*
     * Host initiated processing for a P node
     */

BEGIN
          /*
           * same as for a unique name, except that the
           * request packet must indicate that a
           * group name claim is being made.
           */

          ...
          G = GROUP;
          ...

          /*
           * send packet
           */
           ...


END
```

P-Node Find Name

```
PROCEDURE find_name(name)

    /*
     * Host initiated processing for a P node
     */

BEGIN
        REPEAT
            /*
             * build packet
             */

            ONT = P;
            G = DONT CARE;

            unicast NAME QUERY REQUEST packet;

            /*
             * a NBNS node might send response packet
             */

            IF receive a WACK RESPONSE THEN
                pause(time from TTL field of response);
            ELSE
                pause(UCAST_REQ_RETRY_TIMEOUT);
        UNTIL response packet received OR
            max transmit threshold exceeded

        IF no response packet received THEN
            return failure;
        ELSE
        IF NOT response tid = request tid THEN
            ignore packet;
        ELSE
        CASE packet type OF
        POSITIVE NAME QUERY RESPONSE:
            return success;

        REDIRECT NAME QUERY RESPONSE:

            /*
             * NBNS node wants this end node
             * to use some other NBNS node
             * to resolve the query.
             */

            repeat query with NBNS address
                in the response packet;
        NEGATIVE NAME QUERY RESPONSE:
            return failure;
```

```
        END /* case */
END /* procedure */
```

P-Node Delete_Name

```
PROCEDURE delete_name (name)

    /*
     * Host initiated processing for a P node
     */

BEGIN

          REPEAT

               /*
                * build packet
                */
               ...

               /*
                * send request
                */

               unicast NAME RELEASE REQUEST packet;
               IF receive a WACK RESPONSE THEN
                    pause(time from TTL field of response);
               ELSE
                    pause(UCAST_REQ_RETRY_TIMEOUT);
          UNTIL retransmit count has been exceeded
               or response been received

          IF response has been received THEN
          CASE packet type OF
          POSITIVE NAME RELEASE RESPONSE:
               return success;
          NEGATIVE NAME RELEASE RESPONSE:

               /*
                * NBNS does want node to delete this
                * name !!!
                */

               return failure;
          END /* case */
END /* procedure */
```

P-Node Incoming Packet Processing

Processing initiated by reception of packets at a P node

```
PROCEDURE process_incoming_packet(packet)

    /*
     * Processing initiated by incoming packets at a P node
     */

BEGIN
        /*
         * always ignore UDP broadcast packets
         */

        IF packet was sent as a broadcast THEN
        BEGIN
            ignore packet;
            return;
        END
        CASE packet type of

        NAME CONFLICT DEMAND:
            IF name exists in local name table THEN
                mark name as in conflict;
            return;

        NAME QUERY REQUEST:
            IF name exists in local name table THEN
            BEGIN /* name exists */

                /*
                 * build packet
                 */
                ...

                /*
                 * send response to the IP address and port
                 * number from which the request was received.
                 */

                send POSITIVE NAME QUERY RESPONSE ;
                return;
            END /* exists */
            ELSE
            BEGIN /* does not exist */

                /*
                 * send response to the requestor
                 */

                send NEGATIVE NAME QUERY RESPONSE ;
```

```
                return;
            END /* does not exist */
        NODE STATUS REQUEST:
            /*
             * Name of "*" may be used for force node to
             * divulge status for administrative purposes
             */
            IF name in local name table OR name = "*" THEN
            BEGIN
                /*
                 * Build response packet and
                 * send to requestor node
                 * Send only those names that are
                 * in the same scope as the scope
                 * in the request packet.
                 */

                send NODE STATUS RESPONSE;
            END

        NAME RELEASE REQUEST:
            /*
             * This will be received if the NBNS wants to flush the
             * name from the local name table, or from the local
             * cache.
             */

            IF name exists in the local name table THEN
            BEGIN
                delete name from local name table;
                inform user that name has been deleted;
            END
            ELSE
                IF name has been cached locally THEN
                BEGIN
                      remove entry from cache:
                END

        END /* case */
END /* procedure */
```

P-Node Timer Initiated Processing

Processing initiated by timer expiration.

```
PROCEDURE timer_expired()
    /*
     * Processing initiated by the expiration of a timer on a P node
     */
BEGIN
          /*
           * Send a NAME REFRESH REQUEST for each name which the
           * TTL which has expired.
           */
          REPEAT
               build NAME REFRESH REQUEST packet;
               REPEAT
                    send packet to NBNS;

                    IF receive a WACK RESPONSE THEN
                         pause(time from TTL field of response);
                    ELSE
                         pause(UCAST_REQ_RETRY_TIMEOUT);
               UNTIL response packet is received or
                    retransmit count has been exceeded

               CASE packet type OF
                 POSITIVE NAME REGISTRATION RESPONSE:
                    /* successfully refreshed */
                    reset TTL timer for this name;

                 NEGATIVE NAME REGISTRATION RESPONSE:
                    /*
                     * refused, can't keep name
                     * assume in conflict
                     */
                    mark name as in conflict;
               END /* case */

          UNTIL request sent for all names for which TTL
               has expired
END /* procedure */
```

M-Node Activity

M nodes behavior is similar to that of P nodes with the addition of some B node-like broadcast actions. M node name service proceeds in two steps:

1. Use broadcast UDP based name service. Depending on the operation, goto step 2.
2. Use directed UDP name service.

The following code for M nodes is exactly the same as for a P node, with the exception that broadcast operations are done before P type operation is attempted.

RFC-1002 Protocol Standard for a NetBIOS Service - M-Node Activity

M-Node Add Name

```
PROCEDURE add_name(newname)

    /*
     * Host initiated processing for a M node
     */

BEGIN

          /*
           * check if name exists on the
           * broadcast area
           */
          REPEAT
               /* build packet */

               ....
               broadcast NAME REGISTRATION REQUEST packet;
               pause(BCAST_REQ_RETRY_TIMEOUT);

          UNTIL response packet is received or
               retransmit count has been   exceeded

          IF valid response received THEN
          BEGIN
               /* cannot claim name */

               return failure;
          END

          /*
           * No objections received within the
           * broadcast area.
           * Send request to name server.
           */

          REPEAT
               /*
                * build packet
                */

               ONT = M;
               ...

               unicast NAME REGISTRATION REQUEST packet;

               /*
                * remote NBNS will send response packet
                */

               IF receive a WACK RESPONSE THEN
```

```
            pause(time from TTL field of response);
        ELSE
            pause(UCAST_REQ_RETRY_TIMEOUT);

    UNTIL response packet is received or
        retransmit count has been exceeded

    IF no response packet was received THEN
    BEGIN /* no response */
        /*
         * NBNS is down.   Cannot claim name.
         */

        return failure; /* name cannot be claimed */
    END /* no response */
    ELSE
    BEGIN /* response */
        IF NOT response tid = request tid THEN
        BEGIN
        ignore response packet;
        END
        ELSE
        CASE packet type OF
        POSITIVE NAME REGISTRATION RESPONSE:

            /*
            * name can be added
            */

            adjust refresh timeout value, TTL;
            return success;         /* name can be added */

        NEGATIVE NAME REGISTRATION RESPONSE:
            return failure; /* name cannot be added */

        END-NODE CHALLENGE REGISTRATION REQUEST:
        BEGIN /* end node challenge */

            /*
            * The response packet has in it the
            * address of the presumed owner of the
            * name.   Challenge that owner.
            * If owner either does not
            * respond or indicates that he no longer
            * owns the name, claim the name.
            * Otherwise, the name cannot be claimed.
            *
            */

            REPEAT
            /*
             * build packet
             */
            ...
```

```
/*
 * send packet to address contained in the
 * response packet
 */

unicast NAME QUERY REQUEST packet;

/*
 * remote node may send response packet
 */

pause(UCAST_REQ_RETRY_TIMEOUT);

UNTIL response packet is received or
    retransmit count has been exceeded
IF no response packet is received THEN
BEGIN /* no response */

/*
 * name can be claimed
 */
REPEAT

      /*
       * build packet
       */
       ...

      unicast NAME UPDATE REQUEST to NBNS;

      /*
       * NBNS node will send response packet
       */

      IF receive a WACK RESPONSE THEN
            pause(time from TTL field of response);
ELSE
       pause(UCAST_REQ_RETRY_TIMEOUT);

UNTIL response packet is received or
      retransmit count has been exceeded
IF no response packet received THEN
BEGIN /* no response */

       /*
        * name could not be claimed
        */

       return failure;
END /* no response */
ELSE
CASE packet type OF
POSITIVE NAME REGISTRATION RESPONSE:
       /*
        * add name
```

```
                    */

                   return success;
                NEGATIVE NAME REGISTRATION RESPONSE:

                   /*
                    * you lose   ...
                    */

                   return failure;
                END /* case */
                END /* no response */
                ELSE
                IF NOT response tid = request tid THEN
                BEGIN
                ignore response packet;
                END

                /*
                * received a response to the "challenge"
                * packet
                */

                CASE packet type OF
                POSITIVE NAME QUERY:

                /*
                 * remote node still has name.
                 */

                return failure;
                NEGATIVE NAME QUERY:

                /*
                 * remote node no longer has name
                 */

                return success;
                END /* case */
            END /* end node challenge */
            END /* case */
        END /* response */
END /* procedure */
```

M-Node Add Group Name

```
PROCEDURE add_group_name(newname)

    /*
     * Host initiated processing for a P node
     */

BEGIN
          /*
           * same as for a unique name, except that the
           * request packet must indicate that a
           * group name claim is being made.
           */

          ...
          G = GROUP;
          ...

          /*
           * send packet
           */
           ...
END
```

M-Node Find Name

```
PROCEDURE find_name(name)

    /*
     * Host initiated processing for a M node
     */

BEGIN
          /*
           * check if any node on the broadcast
           * area has the name
           */

          REPEAT
               /* build packet */
               ...

               broadcast NAME QUERY REQUEST packet;
               pause(BCAST_REQ_RETRY_TIMEOUT);
          UNTIL response packet received OR
               max transmit threshold exceeded

          IF valid response received THEN
          BEGIN
               save response as authoritative response;
               start_timer(CONFLICT_TIMER);
               return success;
          END

          /*
           * no valid response on the b'cast segment.
           * Try the name server.
           */

          REPEAT
               /*
                * build packet
                */

               ONT = M;
               G = DONT CARE;

               unicast NAME QUERY REQUEST packet to NBNS;

               /*
                * a NBNS node might send response packet
                */

               IF receive a WACK RESPONSE THEN
                    pause(time from TTL field of response);
               ELSE
```

```
                pause(UCAST_REQ_RETRY_TIMEOUT);
        UNTIL response packet received OR
            max transmit threshold exceeded

        IF no response packet received THEN
            return failure;
        ELSE
        IF NOT response tid = request tid THEN
            ignore packet;
        ELSE
        CASE packet type OF
        POSITIVE NAME QUERY RESPONSE:
            return success;

        REDIRECT NAME QUERY RESPONSE:

            /*
             * NBNS node wants this end node
             * to use some other NBNS node
             * to resolve the query.
             */

             repeat query with NBNS address
                 in the response packet;
        NEGATIVE NAME QUERY RESPONSE:
            return failure;

        END /* case */
END /* procedure */
```

M-Node Delete Name

```
PROCEDURE delete_name (name)

    /*
     * Host initiated processing for a P node
     */

BEGIN
        /*
         * First, delete name on NBNS
         */

        REPEAT

            /*
             * build packet
             */
            ...

            /*
             * send request
             */

            unicast NAME RELEASE REQUEST packet to NBNS;

            IF receive a WACK RESPONSE THEN
                pause(time from TTL field of response);
            ELSE
                pause(UCAST_REQ_RETRY_TIMEOUT);
        UNTIL retransmit count has been exceeded
            or response been received

        IF response has been received THEN
        CASE packet type OF
        POSITIVE NAME RELEASE RESPONSE:
            /*
             * Deletion of name on b'cast segment is deferred
             * until after NBNS has deleted the name
             */

            REPEAT
                /* build packet */

                ...
                broadcast NAME RELEASE REQUEST;
                pause(BCAST_REQ_RETRY_TIMEOUT);
            UNTIL rexmt threshold exceeded

            return success;
        NEGATIVE NAME RELEASE RESPONSE:
```

```
                /*
                 * NBNS does want node to delete this
                 * name
                 */
                return failure;
            END /* case */
END /* procedure */
```

M-Node Incoming Packet Processing

Processing initiated by reception of packets at a M node

```
PROCEDURE process_incoming_packet(packet)

    /*
     * Processing initiated by incoming packets at a M node
     */

BEGIN
        CASE packet type of

        NAME CONFLICT DEMAND:
            IF name exists in local name table THEN
                mark name as in conflict;
            return;

        NAME QUERY REQUEST:
            IF name exists in local name table THEN
            BEGIN /* name exists */

                /*
                 * build packet
                 */
                ...

                /*
                 * send response to the IP address and port
                 * number from which the request was received.
                 */

                send POSITIVE NAME QUERY RESPONSE ;
                return;
            END /* exists */
            ELSE
            BEGIN /* does not exist */

                /*
                 * send response to the requestor
                 */

                IF request NOT broadcast THEN
                      /*
                       * Don't send negative responses to
                       * queries sent by B nodes
                       */
                      send NEGATIVE NAME QUERY RESPONSE ;
                return;
            END /* does not exist */
        NODE STATUS REQUEST:
```

```
            BEGIN
            /*
             * Name of "*" may be used for force node to
             * divulge status for administrative purposes
             */
            IF name in local name table OR name = "*" THEN
                /*
                 * Build response packet and
                 * send to requestor node
                 * Send only those names that are
                 * in the same scope as the scope
                 * in the request packet.
                 */

                send NODE STATUS RESPONSE;
            END

        NAME RELEASE REQUEST:
            /*
             * This will be received if the NBNS wants to flush the
             * name from the local name table, or from the local
             * cache.
             */

            IF name exists in the local name table THEN
            BEGIN
                delete name from local name table;
                inform user that name has been deleted;
            END
            ELSE
                IF name has been cached locally THEN
                BEGIN
                      remove entry from cache:
                END

        NAME REGISTRATION REQUEST (UNIQUE):
            IF name exists in local name table THEN
                send NEGATIVE NAME REGISTRATION RESPONSE ;
        NAME REGISTRATION REQUEST (GROUP):
            IF name exists in local name table THEN
            BEGIN
                IF local entry is a unique name THEN
                    send NEGATIVE NAME REGISTRATION RESPONSE ;
            END
        END /* case */
END /* procedure */
```

M-Node Timer Initiated Processing

Processing initiated by timer expiration:

```
PROCEDURE timer_expired()
    /*
     * Processing initiated by the expiration of a timer on a M node
     */
BEGIN
          /*
           * Send a NAME REFRESH REQUEST for each name which the
           * TTL which has expired.
           */
          REPEAT
               build NAME REFRESH REQUEST packet;
               REPEAT
                    send packet to NBNS;

                    IF receive a WACK RESPONSE THEN
                         pause(time from TTL field of response);
                    ELSE
                         pause(UCAST_REQ_RETRY_TIMEOUT);
               UNTIL response packet is received or
                    retransmit count has been exceeded

               CASE packet type OF
                 POSITIVE NAME REGISTRATION RESPONSE:
                    /* successfully refreshed */
                    reset TTL timer for this name;

                 NEGATIVE NAME REGISTRATION RESPONSE:
                    /*
                     * refused, can't keep name
                     * assume in conflict
                     */
                    mark name as in conflict;
               END /* case */

          UNTIL request sent for all names for which TTL
               has expired
END /* procedure */
```

NBNS Activity

A NBNS node will receive directed packets from P and M nodes. Reply packets are always sent as directed packets to the source IP address and UDP port number. Received broadcast packets must be ignored.

NBNS Incoming Packet Processing
NBNS Timer Initiated Processing

NBNS Incoming Packet Processing

```
PROCEDURE process_incoming_packet(packet)

    /*
     * Incoming packet processing on a NS node
     */

BEGIN
        IF packet was sent as a broadcast THEN
        BEGIN
            discard packet;
            return;
        END
        CASE packet type of

        NAME REGISTRATION REQUEST (UNIQUE):
            IF unique name exists in data base THEN
            BEGIN /* unique name exists */
                /*
                 * NBNS node may be a "passive"
                 * server in that it expects the
                 * end node to do the challenge
                 * server.   Such a NBNS node is
                 * called a "non-secure" server.
                 * A "secure" server will do the
                 * challenging before it sends
                 * back a response packet.
                 */

                IF non-secure THEN
                BEGIN
                    /*
                     * build response packet
                     */
                    ...


                    /*
                     * let end node do the challenge
                     */

                    send END-NODE CHALLENGE NAME REGISTRATION
                        RESPONSE;
                    return;
                END
                ELSE
                /*
                 * secure server - do the name
                 * challenge operation
                 */
                REPEAT
```

```
      send NAME QUERY REQUEST;
      pause(UCAST_REQ_RETRY_TIMEOUT);
UNTIL response has been received or
       retransmit count has been exceeded
IF no response was received THEN
BEGIN

       /* node down */

       update data base - remove entry;
       update data base - add new entry;
       send POSITIVE NAME REGISTRATION RESPONSE;
       return;
END
ELSE
BEGIN /* challenged node replied */
      /*
       * challenged node replied with
       * a response packet
       */

      CASE packet type

      POSITIVE NAME QUERY RESPONSE:

       /*
        * name still owned by the
        * challenged node
        *
        * build packet and send response
        */
        ...


       /*
        * Note: The NBNS will need to
        * keep track (based on transaction id) of
        * the IP address and port number
        * of the original requestor.
        */

       send NEGATIVE NAME REGISTRATION RESPONSE;
       return;
      NEGATIVE NAME QUERY RESPONSE:

       update data base - remove entry;
       update data base - add new   entry;

       /*
        * build response packet and send
        * response
        */
       send POSITIVE NAME REGISTRATION RESPONSE;
       return;
      END /* case */
```

```
        END /* challenged node replied */
    END /* unique name exists in data base */
    ELSE
    IF group name exists in data base THEN
    BEGIN /* group names exists */

        /*
         * Members of a group name are NOT
         * challenged.
         * Make the assumption that
         * at least some of the group members
         * are still alive.
         * Refresh mechanism will
         * allow the NBNS to detect when all
         * members of a group no longer use that
         * name
         */

         send NEGATIVE NAME REGISTRATION RESPONSE;
    END /* group name exists */
    ELSE
    BEGIN /* name does not exist */

        /*
         * Name does not exist in data base
         *
         * This code applies to both non-secure
         * and secure server.
         */

        update data base - add new entry;
        send POSITIVE NAME REGISTRATION RESPONSE;
        return;
    END

NAME QUERY REQUEST:
    IF name exists in data base THEN
    BEGIN
        /*
         * build response packet and send to
         * requestor
         */
         ...

        send POSITIVE NAME QUERY RESPONSE;
        return;
    ELSE
    BEGIN
        /*
         * build response packet and send to
         * requestor
         */
         ...

        send NEGATIVE NAME QUERY RESPONSE;
```

```
        return;
    END

NAME REGISTRATION REQUEST (GROUP):
    IF name exists in data base THEN
    BEGIN
        IF local entry is a unique name THEN
        BEGIN /* local is unique */

            IF non-secure THEN
            BEGIN
             send   END-NODE CHALLENGE NAME
                    REGISTRATION RESPONSE;
             return;
            END

            REPEAT
             send NAME QUERY REQUEST;
             pause(UCAST_REQ_RETRY_TIMEOUT);
            UNTIL response received or
                  retransmit count exceeded
            IF no response received or
                  NEGATIVE NAME QUERY RESPONSE
                   received THEN
            BEGIN
             update data base - remove entry;
             update data base - add new entry;
             send POSITIVE NAME REGISTRATION RESPONSE;
             return;
            END
            ELSE
            BEGIN
             /*
              * name still being held
              * by challenged node
              */

              send NEGATIVE NAME REGISTRATION RESPONSE;
            END
        END /* local is unique */
        ELSE
        BEGIN /* local is group   */
              /*
               * existing entry is a group name
               */

              update data base - remove entry;
              update data base - add new entry;
              send POSITIVE NAME REGISTRATION RESPONSE;
              return;
        END /* local is group */
    END /* names exists */
    ELSE
    BEGIN /* does not exist */
```

```
        /* name does not exist in data base */

        update data base - add new entry;
        send POSITIVE NAME REGISTRATION RESPONSE;
        return;
    END /* does not exist */

NAME RELEASE REQUEST:

    /*
     * secure server may choose to disallow
     * a node from deleting a name
     */

    update data base - remove entry;
    send POSITIVE NAME RELEASE RESPONSE;
    return;

NAME UPDATE REQUEST:

    /*
     * End-node completed a successful challenge,
     * no update database
     */

    IF secure server THEN
        send NEGATIVE NAME REGISTRATION RESPONSE;
    ELSE
    BEGIN /* new entry */
        IF entry already exists THEN
              update data base - remove entry;
        update data base - add new entry;
        send POSITIVE NAME REGISTRATION RESPONSE;
        start_timer(TTL);
    END

NAME REFRESH REQUEST:
    check for consistency;
    IF node not allowed to have name THEN
    BEGIN

        /*
         * tell end node that it can't have name
         */
        send NEGATIVE NAME REGISTRATION RESPONSE;
    END
    ELSE
    BEGIN

        /*
         * send confirmation response to the
         * end node.
         */
        send POSITIVE NAME REGISTRATION;
        start_timer(TTL);
```

```
            END
            return;
        END /* case */
END /* procedure */
```

Nbns Timer Initiated Processing

A NS node uses timers to flush out entries from the data base. Each entry in the data base is removed when its timer expires. This time value is a multiple of the refresh TTL established when the name was registered.

```
PROCEDURE timer_expired()

    /*
     * processing initiated by expiration of TTL for a given name
     */

BEGIN
          /*
           * NBNS can (optionally) ensure
           * that the node is actually down
           * by sending a NODE STATUS REQUEST.
           * If such a request is sent, and
           * no response is received, it can
           * be assumed that the node is down.
           */
          remove entry from data base;
END
```

Session Service Protocols

The following are variables and should be configurable by the NetBIOS user. The default values of these variables is found in "Defined Constants and Variables" in the Detailed Specification.):

- SSN_RETRY_COUNT - The maximum number TCP connection attempts allowable per a single NetBIOS call request.
- SSN_CLOSE_TIMEOUT is the time period to wait when closing the NetBIOS session before killing the TCP connection if session sends are outstanding.

The following are Defined Constants for the NetBIOS Session Service. (See "Defined Constants and Variables" in the Detailed Specification for the value of these constants):

- SSN_SRVC_TCP_PORT - is the globally well-known TCP port allocated for the NetBIOS Session Service. The service accepts TCP connections on this port to establish NetBIOS Sessions. The TCP connection established to this port by the caller is initially used for the exchange of NetBIOS control information. The actual NetBIOS data connection may also pass through this port or, through the retargetting facility, through another port.

User Request Processing

```
PROCEDURE listen(listening name, caller name)
    /*
     * User initiated processing for B, P and M nodes
     *
     * This procedure assumes that an incoming session will be
     * retargetted here by a session server.
     */
BEGIN
          Do TCP listen; /* Returns TCP port used */
          Register listen with Session Service, give names and
               TCP port;

          Wait for TCP connection to open;     /* Incoming call */

          Read SESSION REQUEST packet from connection

          Process session request (see section on
               processing initiated by the reception of session
               service packets);

          Inform Session Service that NetBIOS listen is complete;

          IF session established THEN
               return success and session information to user;
          ELSE
               return failure;
END /* procedure */

PROCEDURE call(calling name, called name)
    /*
     * user initiated processing for B, P and M nodes
     */

    /*
     * This algorithm assumes that the called name is a unique name.
     * If the called name is a group name, the call() procedure
     * needs to cycle through the members of the group
     * until either (retry_count == SSN_RETRY_COUNT) or
     * the list has been exhausted.
     */
BEGIN
          retry_count = 0;
          retarget = FALSE;     /* TRUE: caller is being retargetted */
          name_query = TRUE;    /* TRUE: caller must begin again with */
                                /*          name query. */

          REPEAT
               IF name_query THEN
               BEGIN
```

```
    do name discovery, returns IP address;
    TCP port = SSN_SRVC_TCP_PORT;

    IF name discovery fails THEN
        return failure;
    ELSE
        name_query = FALSE;
END

/*
 * now have IP address and TCP port of
 * remote party.
 */

establish TCP connection with remote party, use an
    ephemeral port as source TCP port;
IF connection refused THEN
BEGIN
    IF retarget THEN
    BEGIN
        /* retry */
        retarget = FALSE;
        use original IP address and TCP port;
        goto LOOP;
    END

    /* retry for just missed TCP listen */

    pause(SESSION_RETRY_TIMER);
    establish TCP connection, again use ephemeral
        port as source TCP port;

    IF connection refused OR
       connection timed out THEN
        return failure;
END
ELSE
IF connection timed out THEN
BEGIN
    IF retarget THEN
    BEGIN
        /* retry */
        retarget = FALSE;
        use original IP address and TCP port;
        goto LOOP;
    END
    ELSE
    BEGIN
        /*
         * incorrect name discovery was done,
         * try again
         */

        inform name discovery process of
            possible error;
```

```
            name_query = TRUE;
            goto LOOP;
      END
END

/*
 * TCP connection has been established
 */

wait for session response packet;
CASE packet type OF

    POSITIVE SESSION RESPONSE:
     return success and session established
            information;

    NEGATIVE SESSION RESPONSE:
    BEGIN
     CASE error OF
        NOT LISTENING ON CALLED NAME:
        NOT LISTENING FOR CALLING NAME:
        BEGIN
          kill TCP connection;
          return failure;
        END

        CALLED NAME NOT PRESENT:
        BEGIN
          /*
           * called name does not exist on
           * remote node
           */

          inform name discovery procedure
                of possible error;

          IF this is a P or M node THEN
          BEGIN
                /*
                 * Inform NetBIOS Name Server
                 * it has returned incorrect
                 * information.
                 */
                send NAME RELEASE REQUEST for called
                  name and IP address to
                  NetBIOS Name Server;
          END
          /* retry from beginning */
          retarget = FALSE;
          name_query = TRUE;
          goto LOOP;
        END /* called name not present */
     END /* case */
    END /* negative response */
```

```
                RETARGET SESSION RESPONSE:
                BEGIN
                 close TCP connection;
                 extract IP address and TCP port from
                       response;
                 retarget = TRUE;
                END /* retarget response */
              END /* case */

LOOP:           retry_count = retry_count + 1;

        UNTIL (retry_count > SSN_RETRY_COUNT);
        return failure;
END /* procedure */
```

Received Packet Processing

These are packets received on a TCP connection before a session has been established. The listen routines attached to a NetBIOS user process need not implement the RETARGET response section. The user process version, separate from a shared Session Service, need only accept (POSITIVE SESSION RESPONSE) or reject (NEGATIVE SESSION RESPONSE) a session request.

```
PROCEDURE session_packet(packet)
    /*
     * processing initiated by receipt of a session service
     * packet for a session in the session establishment phase.
     * Assumes the TCP connection has been accepted.
     */
BEGIN
         CASE packet type

             SESSION REQUEST:
             BEGIN
              IF called name does not exist on node THEN
              BEGIN
                   send NEGATIVE SESSION RESPONSE with CALLED
                         NAME NOT PRESENT error code;
                   close TCP connection;
              END

              Search for a listen with CALLING NAME for CALLED
                   NAME;
              IF matching listen is found THEN
              BEGIN
                   IF port of listener process is port TCP
                       connection is on THEN
                   BEGIN
                         send POSITIVE SESSION RESPONSE;

                         Hand off connection to client process
                               and/or inform user session is
                               established;
                   END
                   ELSE
                   BEGIN
                         send RETARGET SESSION RESPONSE with
                               listener's IP address and
                               TCP port;
                         close TCP connection;
                   END
              END
              ELSE
              BEGIN
                   /* no matching listen pending */
```

```
                send NEGATIVE SESSION RESPONSE with either
                     NOT LISTENING ON CALLED NAME or NOT
                     LISTENING FOR CALLING NAME error
                     code;
                close TCP connection;
             END
            END /* session request */
          END /* case */
END /* procedure */
```

User Request Processing

```
PROCEDURE send_message(user_message)
BEGIN
        build SESSION MESSAGE header;
        send SESSION MESSAGE header;
        send user_message;
        reset and restart keep-alive timer;
        IF send fails THEN
        BEGIN
            /*
             * TCP connection has failed */
             */
            close NetBIOS session;
            inform user that session is lost;
            return failure;
        END
        ELSE
            return success;
END
```

Received Packet Processing

These are packets received after a session has been established.

```
PROCEDURE session_packet(packet)
    /*
     * processing initiated by receipt of a session service
     * packet for a session in the data transfer phase.
     */
BEGIN
         CASE packet type OF

             SESSION   MESSAGE:
             BEGIN
              process message header;
              read in user data;
              reset and restart keep-alive timer;
              deliver data to user;
             END /* session message */

             SESSION KEEP ALIVE:
              discard packet;

         END /* case */
END /* procedure */
```

Processing Initiated By Timer

```
PROCEDURE session_ka_timer()
    /*
     * processing initiated when session keep alive timer expires
     */
BEGIN
                send SESSION KEEP ALIVE, if configured;
                IF send fails THEN
                BEGIN
                /* remote node, or path to it, is down */

                abort TCP connection;
                close NetBIOS session;
                inform user that session is lost;
                return;
                END
END /* procedure */
```

User Request Processing

```
PROCEDURE close_session()

    /* initiated by a user request to close a session */

BEGIN
        close gracefully the TCP connection;

        WAIT for the connection to close or SSN_CLOSE_TIMEOUT
            to expire;

        IF time out expired THEN
            abort TCP connection;
END /* procedure */
```

Reception Indication Processing

```
PROCEDURE close_indication()
    /*
     * initiated by a TCP indication of a close request from
     * the remote connection partner.
     */
BEGIN
        close gracefully TCP connection;

        close NetBIOS session;

        inform user session closed by remote partner;
END /* procedure */
```

NetBIOS Datagram Service Protocols

The following are GLOBAL variables and should be NetBIOS user configurable:

- SCOPE_ID: the non-leaf section of the domain name preceded by a '.' which represents the domain of the NetBIOS scope for the NetBIOS name. The following protocol description only supports single scope operation.
- MAX_DATAGRAM_LENGTH: the maximum length of an IP datagram. The minimal maximum length defined in for IP is 576 bytes. This value is used when determining whether to fragment a NetBIOS datagram. Implementations are expected to be capable of receiving unfragmented NetBIOS datagrams up to their maximum size.
- BROADCAST_ADDRESS: the IP address B-nodes use to send datagrams with group name destinations and broadcast datagrams. The default is the IP broadcast address for a single IP network.

The following are Defined Constants for the NetBIOS Datagram Service:

- DGM_SRVC_UDP_PORT: the globally well-known UDP port allocated where the NetBIOS Datagram Service receives UDP packets. See section 6, "Defined Constants", for its value.

B Node Transmission of NetBIOS Datagrams

```
PROCEDURE send_datagram(data, source, destination, broadcast)

    /*
     * user initiated processing on B node
     */

BEGIN
        group = FALSE;

        do name discovery on destination name, returns name type and
            IP address;
        IF name type is group name THEN
        BEGIN
            group = TRUE;
        END

        /*
         * build datagram service UDP packet;
         */
        convert source and destination NetBIOS names into
            half-ASCII, biased encoded name;
        SOURCE_NAME = cat(source, SCOPE_ID);
        SOURCE_IP = this nodes IP address;
        SOURCE_PORT =   DGM_SRVC_UDP_PORT;

        IF NetBIOS broadcast THEN
        BEGIN
            DESTINATION_NAME = cat("*", SCOPE_ID)
        END
        ELSE
        BEGIN
            DESTINATION_NAME = cat(destination, SCOPE_ID)
        END

        MSG_TYPE = select_one_from_set
            {BROADCAST, DIRECT_UNIQUE, DIRECT_GROUP}
        DGM_ID = next transaction id for Datagrams;
        DGM_LENGTH = length of data + length of second level encoded
            source and destination names;

        IF (length of the NetBIOS Datagram, including UDP and
            IP headers, > MAX_DATAGRAM_LENGTH) THEN
        BEGIN
            /*
             * fragment NetBIOS datagram into 2 UDP packets
             */
            Put names into 1st UDP packet and any data that fits
                after names;
            Set MORE and FIRST bits in 1st UDP packet's FLAGS;
            OFFSET in 1st UDP = 0;
```

```
            Replicate NetBIOS Datagram header from 1st UDP packet
                into 2nd UDP packet;
            Put rest of data in 2nd UDP packet;
            Clear MORE and FIRST bits in 2nd UDP packet's FLAGS;
            OFFSET in 2nd UDP = DGM_LENGTH - number of name and
                data bytes in 1st UDP;
        END
        BEGIN
            /*
             * Only need one UDP packet
             */
            USER_DATA = data;
            Clear MORE bit and set FIRST bit in FLAGS;
            OFFSET = 0;
        END

        IF (group == TRUE) OR (NetBIOS broadcast) THEN
        BEGIN
            send UDP packet(s) to BROADCAST_ADDRESS;
        END
        ELSE
        BEGIN
            send UDP packet(s) to IP address returned by name
                discovery;
        END
END /* procedure */
```

P and M Node Transmission of NetBIOS Datagrams

```
PROCEDURE send_datagram(data, source, destination, broadcast)

    /*
     * User initiated processing on P and M node.
     *
     * This processing is the same as for B nodes except for
     * sending broadcast and multicast NetBIOS datagrams.
     */

BEGIN
        group = FALSE;

        do name discovery on destination name, returns name type
            and IP address;
        IF name type is group name THEN
        BEGIN
            group = TRUE;
        END

        /*
         * build datagram service UDP packet;
         */
        convert source and destination NetBIOS names into
            half-ASCII, biased encoded name;
        SOURCE_NAME = cat(source, SCOPE_ID);
        SOURCE_IP = this nodes IP address;
        SOURCE_PORT =   DGM_SRVC_UDP_PORT;

        IF NetBIOS broadcast THEN
        BEGIN
            DESTINATION_NAME = cat("*", SCOPE_ID)
        END
        ELSE
        BEGIN
            DESTINATION_NAME = cat(destination, SCOPE_ID)
        END

        MSG_TYPE = select_one_from_set
            {BROADCAST, DIRECT_UNIQUE, DIRECT_GROUP}
        DGM_ID = next transaction id for Datagrams;
        DGM_LENGTH = length of data + length of second level encoded
            source and destination names;

        IF (length of the NetBIOS Datagram, including UDP and
            IP headers, > MAX_DATAGRAM_LENGTH) THEN
        BEGIN
            /*
             * fragment NetBIOS datagram into 2 UDP packets
             */
            Put names into 1st UDP packet and any data that fits
```

```
            after names;
        Set MORE and FIRST bits in 1st UDP packet's FLAGS;

        OFFSET in 1st UDP = 0;

        Replicate NetBIOS Datagram header from 1st UDP packet
            into 2nd UDP packet;
        Put rest of data in 2nd UDP packet;
        Clear MORE and FIRST bits in 2nd UDP packet's FLAGS;
        OFFSET in 2nd UDP = DGM_LENGTH - number of name and
            data bytes in 1st UDP;
    END
    BEGIN
        /*
         * Only need one UDP packet
         */
        USER_DATA = data;
        Clear MORE bit and set FIRST bit in FLAGS;
        OFFSET = 0;
    END

    IF (group == TRUE) OR (NetBIOS broadcast) THEN
    BEGIN
        /*
         * Sending of following query is optional.
         * Node may send datagram to NBDD immediately
         * but NBDD may discard the datagram.
         */
        send DATAGRAM QUERY REQUEST to NBDD;
        IF response is POSITIVE QUERY RESPONSE THEN
            send UDP packet(s) to NBDD Server IP address;
        ELSE
        BEGIN
            get list of destination nodes from NBNS;
            FOR EACH node in list
            BEGIN
                send UDP packet(s) to this node's
                    IP address;
            END
        END
    END
    ELSE
    BEGIN
        send UDP packet(s) to IP address returned by name
            discovery;
END /* procedure */
```

Reception of NetBIOS Datagrams by All Nodes

The following algorithm discards out of order NetBIOS Datagram fragments. An implementation which reassembles out of order NetBIOS Datagram fragments conforms to this specification. The fragment discard timer is initialized to the value FRAGMENT_TO. This value should be user configurable. The default value is given in Section 6, "Defined Constants and Variables".

```
PROCEDURE datagram_packet(packet)

    /*
     * processing initiated by datagram packet reception
     * on B, P and M nodes
     */
BEGIN
          /*
           * if this node is a P node, ignore
           * broadcast packets.
           */

          IF this is a P node AND incoming packet is
               a broadcast packet THEN
          BEGIN
               discard packet;
          END

          CASE packet type OF

              DATAGRAM SERVICE:
              BEGIN
               IF FIRST bit in FLAGS is set THEN
               BEGIN
                    IF MORE bit in FLAGS is set THEN
                    BEGIN
                         Save 1st UDP packet of the Datagram;
                         Set this Datagram's fragment discard
                         timer to FRAGMENT_TO;
                         return;
                    END
                    ELSE
                         Datagram is composed of a single
                           UDP packet;
               END
               ELSE
               BEGIN
                    /* Have the second fragment of a Datagram */

                    Search for 1st fragment by source IP address
                        and DGM_ID;
                    IF found 1st fragment THEN
                         Process both UDP packets;
```

```
    ELSE
    BEGIN
        discard 2nd fragment UDP packet;
        return;
    END
END

IF DESTINATION_NAME is '*' THEN
BEGIN
    /* NetBIOS broadcast */

    deliver USER_DATA from UDP packet(s) to all
        outstanding receive broadcast
        datagram requests;
    return;
END
ELSE
BEGIN /* non-broadcast */
    /* Datagram for Unique or Group Name */

    IF DESTINATION_NAME is not present in the
       local name table THEN
    BEGIN
        /* destination not present */
        build DATAGRAM ERROR packet, clear
        FIRST and MORE bit, put in
        this nodes IP and PORT, set
        ERROR_CODE;
        send DATAGRAM ERROR packet to source IP address
        and port of UDP;
        discard UDP packet(s);
        return;
    END
    ELSE
    BEGIN /* good */
        /*
         * Replicate received NetBIOS datagram for
         * each recipient
         */
        FOR EACH pending NetBIOS user's receive datagram operation
        BEGIN
            IF source name of operation matches destination name of packet
            THEN
            BEGIN
                deliver USER_DATA from UDP packet(s);
                    END
        END /* for each */
        return;
    END /* good */
END /* non-broadcast */
 END /* datagram service */

DATAGRAM ERROR:
BEGIN
    /*
```

```
                * name service returned incorrect information
                */

            inform local name service that incorrect
            information was provided;

            IF this is a P or M node THEN
            BEGIN
                    /*
                     * tell NetBIOS Name Server that it may
                     * have given incorrect information
                     */

                    send NAME RELEASE REQUEST with name
                    and incorrect IP address to NetBIOS
                    Name Server;
                END
            END /* datagram error */

        END /* case */
END
```

Protocols for the NBDD

The key to NetBIOS Datagram forwarding service is the packet delivered to the destination end node must have the same NetBIOS header as if the source end node sent the packet directly to the destination end node. Consequently, the NBDD does not reassemble NetBIOS Datagrams. It forwards the UDP packet as is.

```
PROCEDURE   datagram_packet(packet)
   /*
    * processing initiated by a incoming datagram service
    * packet on a NBDD node.
    */

BEGIN
        CASE packet type OF

           DATAGRAM SERVICE:
           BEGIN
               IF packet was sent as a directed
                 NetBIOS datagram THEN
               BEGIN
                /*
                 * provide group forwarding service
                 *
                 * Forward datagram to each member of the
                 * group.   Can forward via:
                 *     1) get list of group members and send
                 *     the DATAGRAM SERVICE packet unicast
                 *     to each
                 *     2) use Group Multicast, if available
                 *     3) combination of 1) and 2)
                 */

                ...

               END

               ELSE
               BEGIN
                /*
                 * provide broadcast forwarding service
                 *
                 * Forward datagram to every node in the
                 * NetBIOS scope.   Can forward via:
                 *     1) get list of group members and send
                 *     the DATAGRAM SERVICE packet unicast
                 *     to each
                 *     2) use Group Multicast, if available
                 *     3) combination of 1) and 2)
                 */

                ...
```

```
            END
        END /* datagram service */

        DATAGRAM ERROR:
        BEGIN
         /*
          * Should never receive these because Datagrams
          * forwarded have source end node IP address and
          * port in NetBIOS header.
          */

         send DELETE NAME REQUEST with incorrect name and
              IP address to NetBIOS Name Server;

        END /* datagram error */

        DATAGRAM QUERY REQUEST:
        BEGIN
         IF can send packet to DESTINATION_NAME THEN
         BEGIN
              /*
               * NBDD is able to relay Datagrams for
               * this name
               */

              send POSITIVE DATAGRAM QUERY RESPONSE to
                REQUEST source IP address and UDP port
                with request's DGM_ID;
         END
         ELSE
         BEGIN
              /*
               * NBDD is NOT able to relay Datagrams for
               * this name
               */

              send NEGATIVE DATAGRAM QUERY RESPONSE to
                REQUEST source IP address and UDP port

                with request's DGM_ID;
         END
        END /* datagram query request */

    END /* case */
END /* procedure */
```

Defined Constants and Variables

RFC-1002 Protocol Standard for a NetBIOS Service - Defined Constants

General Constants

SCOPE_ID The name of the NetBIOS scope.

This is expressed as a character string meeting the requirements of the domain name system and without a leading or trailing "dot".

An implementation may elect to make this a single global value for the node or allow it to be specified with each separate NetBIOS name (thus permitting cross-scope references.)

BROADCAST_ADDRESS

An IP address composed of the nodes's network and subnetwork numbers with all remaining bits set to one.

I.e. "Specific subnet" broadcast addressing according to section 2.3 of RFC 950.

BCAST_REQ_RETRY_TIMEOUT 250 milliseconds.

An adaptive timer may be used.

BCAST_REQ_RETRY_COUNT 3

UCAST_REQ_RETRY_TIMEOUT 5 seconds

An adaptive timer may be used.

UCAST_REQ_RETRY_COUNT 3

MAX_DATAGRAM_LENGTH 576 bytes (default)

RFC-1002 Protocol Standard for a NetBIOS Service - Defined Constants

Name Service Constants

REFRESH_TIMER	Negotiated with NBNS for each name.
CONFLICT_TIMER	1 second
	Implementations may chose a longer value.
NAME_SERVICE_TCP_PORT	137 (decimal)
NAME_SERVICE_UDP_PORT	137 (decimal)
INFINITE_TTL	0

Session Service Constants

SSN_SRVC_TCP_PORT 139 (decimal)

SSN_RETRY_COUNT 4 (default)

Re-configurable by user.

SSN_CLOSE_TIMEOUT 30 seconds (default)

Re-configurable by user.

SSN_KEEP_ALIVE_TIMEOUT 60 seconds, recommended.

May be set to a higher value. (Session keep-alives are used only if configured.)

RFC-1002 Protocol Standard for a NetBIOS Service - Defined Constants

Datagram Service Constants

DGM_SRVC_UDP_PORT	138 (decimal)
FRAGMENT_TO	2 seconds (default)

REFERENCES

[1] "Protocol Standard For a NetBIOS Service on a TCP/UDP Transport: Concepts and Methods", RFC 1001, March 1987.

[2] J. Reynolds, J. Postel, "Assigned Numbers", RFC 990, November 1986.

[3] P. Mockapetris, "Domain Names - Implementation and Specification", RFC 883, November 1983.

NAME_TRN_ID	Transaction ID for Name Service Transaction. Requestor places a unique value for each active transaction. Responder puts NAME_TRN_ID value from request packet in response packet.

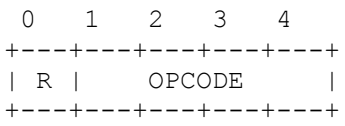
RCODE	Result codes of request.	Table of RCODE values for each response packet below.

QDCOUNT Unsigned 16 bit integer specifying the number of entries in the question section of a Name Service packet. Always zero (0) for responses. Must be non-zero for all NetBIOS Name requests.

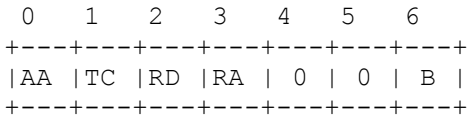
ANCOUNT Unsigned 16 bit integer specifying the number of resource records in the answer section of a Name Service packet.

NSCOUNT Unsigned 16 bit integer specifying the number of resource records in the authority section of a Name Service packet.

ARCOUNT	Unsigned 16 bit integer specifying the number of resource records in the additional records section of a Name Service packet.



Symbol	Bit(s)	Description
OPCODE	1-4	Operation specifier: 0 = query 5 = registration 6 = release 7 = WACK 8 = refresh
R	0	RESPONSE flag: if bit == 0 then request packet if bit == 1 then response packet.



Symbol	Bit(s)	Description
B	6	Broadcast Flag.
RA	3	Recursion Available Flag.
RD	2	Recursion Desired Flag.
TC	1	Truncation Flag.
AA	0	Authoritative Answer flag.

RR_NAME The compressed name representation of the NetBIOS name corresponding to this resource record.

QUESTION_NAME	The compressed name representation of the NetBIOS name for the request.

NB 0x0020 NetBIOS general Name Service Resource Record

NBSTAT 0x0021 NetBIOS NODE STATUS Resource Record (See NODE STATUS REQUEST)

B Broadcast Flag.
= 1: packet was broadcast or multicast
= 0: unicast

IN	0x0001	Internet class

Symbol	Bit(s)	Description:
RESERVED	3-15	Reserved for future use. Must be zero (0).
ONT	1,2	Owner Node Type: 00 = B node 01 = P node 10 = M node 11 = Reserved for future use For registration requests this is the claimant's type. For responses this is the actual owner's type.
G	0	Group Name Flag. If one (1) then the RR_NAME is a GROUP NetBIOS name. If zero (0) then the RR_NAME is a UNIQUE NetBIOS name.

The NB_ADDRESS field of the RESOURCE RECORD RDATA field for RR_TYPE of "NB" is the IP address of the name's owner.

TTL	The Time To Live of a the resource record's name.

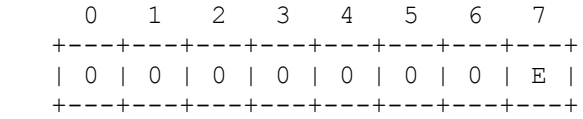
The NB_ADDRESS field of the RESOURCE RECORD RDATA field for RR_TYPE of "NB" is the IP address of the name's owner.

ONT Owner Node Type:
00 = B node
01 = P node
10 = M node
11 = Reserved for future use
For registration requests this is the claimant's type.
For responses this is the actual owner's type.

Session Packet Types (in hexidecimal):

- 00 - SESSION MESSAGE
- 81 - SESSION REQUEST
- 82 - POSITIVE SESSION RESPONSE
- 83 - NEGATIVE SESSION RESPONSE
- 84 - RETARGET SESSION RESPONSE
- 85 - SESSION KEEP ALIVE

Bit definitions of the Session Packet FLAGS field:

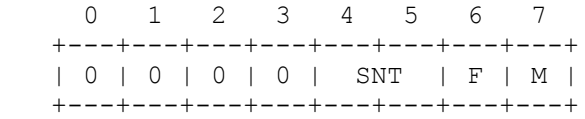


Symbol	Bit(s)	Description
E	7	Length extension, used as an additional, high-order bit on the LENGTH field.
RESERVED	0-6	Reserved, must be zero (0)

MSG_TYPE values (in hexidecimal):

10 - DIRECT_UNIQUE DATAGRAM
11 - DIRECT_GROUP DATAGRAM
12 - BROADCAST DATAGRAM
13 - DATAGRAM ERROR
14 - DATAGRAM QUERY REQUEST
15 - DATAGRAM POSITIVE QUERY RESPONSE
16 - DATAGRAM NEGATIVE QUERY RESPONSE

Bit definitions of the FLAGS field:



Symbol	Bit(s)	Description
M	7	MORE flag, If set then more NetBIOS datagram fragments follow.
F	6	FIRST packet flag, If set then this is first (and possibly only) fragment of NetBIOS datagram
SNT	4,5	Source End-Node type: 00 = B node 01 = P node 10 = M node 11 = NBDD
RESERVED	0-3	Reserved, must be zero (0)

RFC-1006 ISO Transport Service on top of the TCP

Version 3

Marshall T. Rose & Dwight E. Cass
May 1987

Status of this Memo

This memo specifies a standard for the Internet community. Hosts on the Internet that choose to implement ISO transport services on top of the TCP are expected to adopt and implement this standard. TCP port 102 is reserved for hosts which implement this standard. Distribution of this memo is unlimited.

This memo specifies version 3 of the protocol and supersedes [RFC983]. Changes between the protocol as described in Request for Comments 983 and this memo are minor, but are unfortunately incompatible.

RFC-1006 ISO Transport on top of the TCP

Introduction and Philosophy

The Internet community has a well-developed, mature set of transport and internetwork protocols (TCP/IP), which are quite successful in offering network and transport services to end-users. The CCITT and the ISO have defined various session, presentation, and application recommendations which have been adopted by the international community and numerous vendors. To the largest extent possible, it is desirable to offer these higher level directly in the ARPA Internet, without disrupting existing facilities. This permits users to develop expertise with ISO and CITT applications which previously were not available in the ARPA Internet. It also permits a more graceful convergence and transition strategy from TCP/IP-based networks to ISO-based networks in the medium-and long-term.

There are two basic approaches which can be taken when "porting" an ISO or CCITT application to a TCP/IP environment. One approach is to port each individual application separately, developing local protocols on top of the TCP. Although this is useful in the short-term (since special-purpose interfaces to the TCP can be developed quickly), it lacks generality.

A second approach is based on the observation that both the ARPA Internet protocol suite and the ISO protocol suite are both layered systems (though the former uses layering from a more pragmatic perspective). A key aspect of the layering principle is that of layer-independence. Although this section is redundant for most readers, a slight bit of background material is necessary to introduce this concept.

Externally, a layer is defined by two definitions:

> a service-offered definition, which describes the services provided by the layer and the interfaces it provides to access those services; and,
>
> a service-required definitions, which describes the services used by the layer and the interfaces it uses to access those services.

Collectively, all of the entities in the network which co-operate to provide the service are known as the service-provider. Individually, each of these entities is known as a service-peer.

Interally, a layer is defined by one definition:

> a protocol definition, which describes the rules which each service-peer uses when communicating with other service-peers.

Putting all this together, the service-provider uses the protocol and services from the layer below to offer the its service to the layer above. Protocol verification, for instance, deals with proving that this in fact happens (and is also a fertile field for many Ph.D. dissertations in computer science).

The concept of layer-independence quite simply is:

> IF one preserves the services offered by the service-provider
>
> THEN the service-user is completely naive with respect to the protocol which the service-peers use

For the purposes of this memo, we will use the layer-independence to define a Transport Service Access Point (TSAP) which appears to be identical to the services and interfaces offered by the ISO/CCITT TSAP (as defined in [ISO8072]), but we will in fact implement theISO TP0 protocol on top of TCP/IP (as defined in [RFC793, RFC791]), not on top of the the ISO/CCITT network protocol. Since the transport class 0 protocol is used over the TCP/IP

connection, it achieves identical functionality as transport class 4. Hence, ISO/CCITT higher level layers (all session, presentation, and application entities) can operate fully without knowledge of the fact that they are running on a TCP/IP internetwork.

RFC-1006 ISO Transport on top of the TCP

Motivation

In migrating from the use of TCP/IP to the ISO protocols, there are several strategies that one might undertake. This memo was written with one particular strategy in mind.

The particular migration strategy which this memo uses is based on the notion of gatewaying between the TCP/IP and ISO ptotocol suites at the transport layer. There are two strong arguments for this approach:

1. Experience teaches us that it takes just as long to get good implementations of the lower level protocols as it takes to get implementations of the higher level ones. In particular, it has been observed that there is still a lot of work being done at the ISO network and transport layers. As a result, implementations of protocols above these layers are not being aggressively pursued. Thus, something must be done "now" to provide a medium in which the higher level protocols can be developed. Since TCP/IP is mature, and essentially provides identical functionality, it is an ideal medium to support this development.
2. Implementation of gateways at the IP and ISO IP layers are probably not of general use in the long term. In effect, this would require each Internet host to support both TP4 and TCP. As such, a better strategy is to implement a graceful migration path from TCP/IP to ISO protocols for the ARPA Internet when the ISO protocols have matured sufficiently.

Both of these arguments indicate that gatewaying should occur at or above the transport layer service access point. Further, the first argument suggests that the best approach is to perform the gatewaying exactly AT the transport service access point to maximize the number of ISO layers which can be developed.

> **Note:**
> This memo does not intend to act as a migration or intercept document. It is intended **only** to meet the needs discussed above. However, it would not be unexpected that the protocol described in this memo might form part of an overall transition plan. The description of such a plan however is **completely** beyond the scope of this memo.

Finally, in general, building gateways between other layers in the TCP/IP and ISO protocol suites is problematic, at best.

To summarize: the primary motivation for the standard described in this memo is to facilitate the process of gaining experience with higher-level ISO protocols (session, presentation, and application). The stability and maturity of TCP/IP are ideal for providing solid transport services independent of actual implementation.

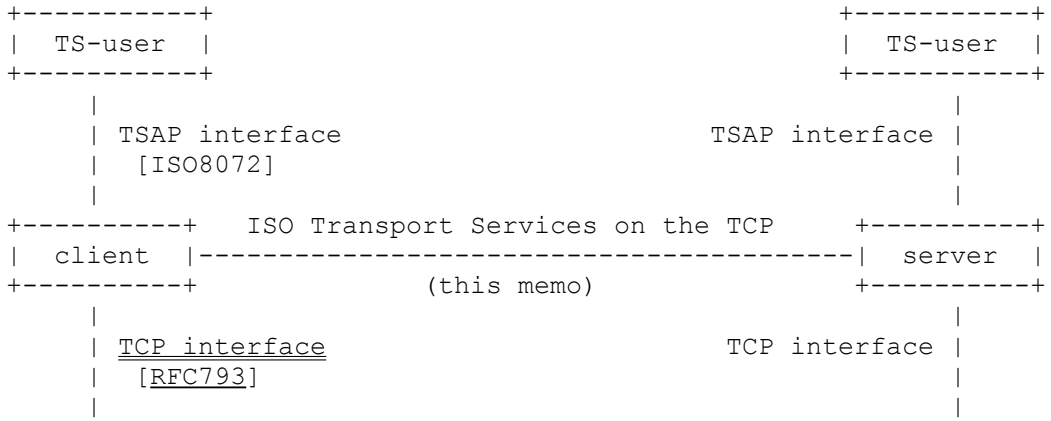
RFC-1006 ISO Transport on top of the TCP

The Model

The [ISO8072] standard describes the ISO transport service definition, henceforth called TP.

> **Aside:**
> This memo references the ISO specifications rather than the CCITT recommendations. The differences between these parallel standards are quite small, and can be ignored, with respect to this memo, without loss of generality. To provide the reader with the relationships:
>
> | | | |
> |---|---|---|
> | Transport service | [ISO8072] | [X.214] |
> | Transport protocol | [ISO8073] | [X.224] |
> | Session protocol | [ISO8327] | [X.225] |

The ISO transport service definition describes the services offered by the TS-provider (transport service) and the interfaces used to access those services. This memo focuses on how the ARPA Transmission Control Protocol (TCP) [RFC793] can be used to offer the services and provide the interfaces.



For expository purposes, the following abbreviations are used:

TS-peer	a process which implements the protocol described by this memo
TS-user	a process talking using the services of a TS-peer
TS-provider	the black-box entity implementing the protocol described by this memo

For the purposes of this memo, which describes version 2 of the TSAP protocol, all aspects of [ISO8072] are supported with one exception:

Quality of Service parameters

In the spirit of CCITT, this is left "for further study". A future version of the protocol will most likely support the QOS parameters for TP by mapping these onto various TCP parameters.

The ISO standards do not specify the format of a session port (termed a TSAP ID). This memo mandates the use of the GOSIP specification [GOSIP86] for the interpretation of this field. (Please refer to Section 5.2, entitled "UPPER LAYERS ADDRESSING".)

Finally, the ISO TSAP is fundamentally symmetric in behavior. There is no underlying client/server model. Instead of a server listening on a well-known port, when a connection is established, the TS-provider generates an INDICATION event which, presumably the TS-

user catches and acts upon. Although this might be implemented by having a server "listen" by hanging on the INDICATION event, from the perspective of the ISO TSAP, all TS-users just sit around in the IDLE state until they either generate a REQUEST or accept an INDICATION.

RFC-1006 ISO Transport on top of the TCP

The Primitives

The protocol assumes that the TCP[RFC793] offers the following service primitives:

Events

connected	- open succeeded (either ACTIVE or PASSIVE)
connect fails	- ACTIVE open failed
data ready	- data can be read from the connection
errored	- the connection has errored and is now closed
closed	- an orderly disconnection has started

Actions

listen on port	- PASSIVE open on the given port
open port	- ACTIVE open to the given port
read data	- data is read from the connection
send data	- data is sent on the connection
close	- the connection is closed (pending data is sent)

This memo describes how to use these services to emulate the following service primitives, which are required by [ISO8073]:

Events

N-CONNECT.INDICATION
- An NS-user (responder) is notified that connection establishment is in progress

N-CONNECT.CONFIRMATION
- An NS-user (responder) is notified that the connection has been established

N-DATA.INDICATION
- An NS-user is notified that data can be read from the connection

N-DISCONNECT.INDICATION
- An NS-user is notified that the connection is closed

Actions

N-CONNECT.REQUEST
- An NS-user (initiator) indicates that it wants to establish a connection

N-CONNECT.RESPONSE
- An NS-user (responder) indicates that it will honor the request

N-DATA.REQUEST
- An NS-user sends data

N-DISCONNECT.REQUEST
- An NS-user indicates that the connection is to be closed

The protocol offers the following service primitives, as defined in [ISO8072], to the TS-user:

Events

T-CONNECT.INDICATION
- a TS-user (responder) is notified that connection establishment is in progress

T-CONNECT.CONFIRMATION
- a TS-user (initiator) is notified that the connection has been established

T-DATA.INDICATION
- a TS-user is notified that data can be read from the connection

T-EXPEDITED DATA.INDICATION
- a TS-user is notified that "expedited" data can be read from the connection

T-DISCONNECT.INDICATION
- a TS-user is notified that the connection is closed

Actions

T-CONNECT.REQUEST
- a TS-user (initiator) indicates that it wants to establish a connection

T-CONNECT.RESPONSE
- a TS-user (responder) indicates that it will honor the request

T-DATA.REQUEST
- a TS-user sends data

T-EXPEDITED DATA.REQUEST
- a TS-user sends "expedited" data

T-DISCONNECT.REQUEST
- a TS-user indicates that the connection is to be closed

RFC-1006 ISO Transport on top of the TCP

The Protocol

The protocol specified by this memo is identical to the protocol for ISO transport class 0, with the following exceptions:

- for testing purposes, initial data may be exchanged during connection establishment
- for testing purposes, an expedited data service is supported
- for performance reasons, a much larger TSDU size is supported
- the network service used by the protocol is provided by the TCP

The ISO transport protocol exchanges information between peers in discrete units of information called transport protocol data units (TPDUs). The protocol defined in this memo encapsulates these TPDUs in discrete units called TPKTs. The structure of these TPKTs and their relationship to TPDUs are discussed in the next section.

Primitives

The mapping between the TCP service primitives and the service primitives expected by transport class 0 are quite straight- forward:

network service	**TCP**
CONNECTION ESTABLISHMENT	
N-CONNECT.REQUEST	open completes
N-CONNECT.INDICATION	listen (PASSIVE open) finishes
N-CONNECT.RESPONSE	listen completes
N-CONNECT.CONFIRMATION	open (ACTIVE open) finishes
DATA TRANSFER	
N-DATA.REQUEST	send data
N-DATA.INDICATION	data ready followed by read data
CONNECTION RELEASE	
N-DISCONNECT.REQUEST	close
N-DISCONNECT.INDICATION	connection closes or errors

Mapping parameters is also straight-forward:

network service	**TCP**
CONNECTION ESTABLISHMENT	
Called address	server's IP address (4 octets)
Calling address	client's IP address (4 octets)
all others	ignored
DATA TRANSFER	
NS-user data (NSDU)	data
CONNECTION RELEASE	
all parameters	ignored

CONNECTION ESTABLISHMENT

The elements of procedure used during connection establishment are identical to those presented in [ISO8073], with three exceptions.

In order to facilitate testing, the connection request and connection confirmation TPDUs may exchange initial user data, using the user data fields of these TPDUs.

In order to experiment with expedited data services, the connection request and connection confirmation TPDUs may negotiate the use of expedited data transfer using the negotiation mechanism specified in [ISO8073] is used (e.g., setting the "use of transport expedited data transfer service" bit in the "Additional Option Selection" variable part). The default is not to use the transport expedited data transfer service.

In order to achieve good performance, the default TPDU size is 65531 octets, instead of 128 octets. In order to negotiate a smaller (standard) TPDU size, the negotiation mechanism specified in [ISO8073] is used (e.g., setting the desired bit in the "TPDU Size" variable part).

To perform an N-CONNECT.REQUEST action, the TS-peer performs an active open to the desired IP address using TCP port 102. When the TCP signals either success or failure, this results in an N-CONNECT.INDICATION action.

To await an N-CONNECT.INDICATION event, a server listens on TCP port 102. When a client successfully connects to this port, the event occurs, and an implicit N-CONNECT.RESPONSE action is performed.

> **Note:**
> In most implementations, a single server will perpetually LISTEN on port 102, handing off connections as they are made

DATA TRANSFER

The elements of procedure used during data transfer are identical to those presented in [ISO8073], with one exception: expedited data may be supported (if so negotiated during connection establishment) by sending a modified ED TPDU (described below). The TPDU is sent on the same TCP connection as all of the other TPDUs. This method, while not faithful to the spirit of [ISO8072], is true to the letter of the specification.

To perform an N-DATA.REQUEST action, the TS-peer constructs the desired TPKT and uses the TCP send data primitive.

To trigger an N-DATA.INDICATION action, the TCP indicates that data is ready and a TPKT is read using the TCP read data primitive.

CONNECTION RELEASE

To perform an N-DISCONNECT.REQUEST action, the TS-peer simply closes the TCP connection.

If the TCP informs the TS-peer that the connection has been closed or has errored, this indicates an N-DISCONNECT.INDICATION event.

RFC-1006 ISO Transport on top of the TCP

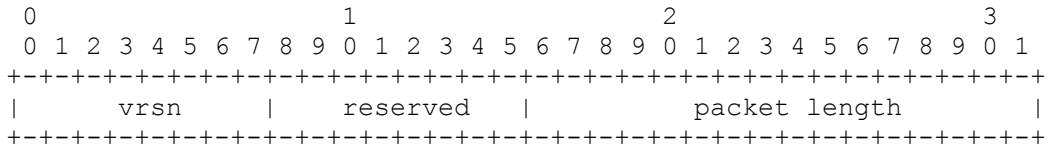
Packet Format

A fundamental difference between the TCP and the network service expected by TP0 is that the TCP manages a continuous stream of octets, with no explicit boundaries. The TP0 expects information to be sent and delivered in discrete objects termed network service data units (NSDUs). Although other classes of transport may combine more than one TPDU inside a single NSDU, transport class 0 does not use this facility. Hence, an NSDU is identical to a TPDU for the purposes of our discussion.

The protocol described by this memo uses a simple packetization scheme in order to delimit TPDUs. Each packet, termed a TPKT, is viewed as an object composed of an integral number of octets, of variable length.

> **Note:**
> For the purposes of presentation, these objects are shown as being 4 octets (32 bits wide). This representation is an artifact of the style of this memo and should nto be interpreted as requiring that a TPKT be a multiple of 4 octets in length.

A TPKT consists of two parts: a packet-header and a TPDU. The format of the header is constant regardless of the type of packet. The format of the packet-header is as follows:



where:

vrsn 8 bits

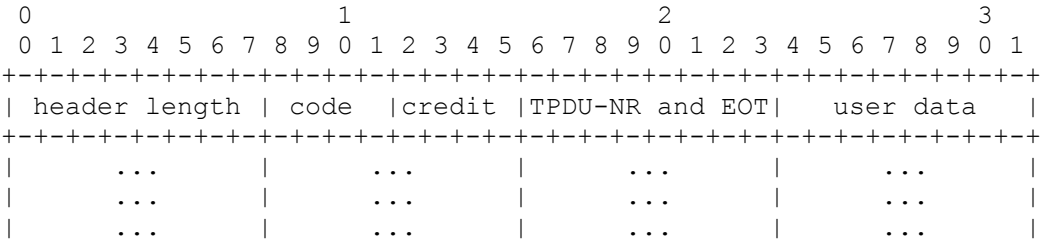
This field is always 3 for the version of the protocol described in this memo.

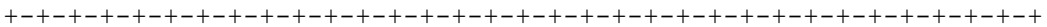
packet length 16 bits (min=7, max=65535)

This field contains the length of entire packet in octets, including packet-header. This permits a maximum TPDU size of 65531 octets. Based on the size of the data transfer (DT) TPDU, this permits a maximum TSDU size of 65524 octets.

The format of the TPDU is defined in [ISO8073]. Note that only TPDUs formatted for transport class 0 are exchanged (different transport classes may use slightly different formats).

To support expedited data, a non-standard TPDU, for expedited data is permitted. The format used for the ED TPDU is nearly identical to the format for the normal data, DT, TPDU. The only difference is that the value used for the TPDU's code is ED, not DT:





After the credit field (which is always ZERO on output and ignored on input), there is one additional field prior to the user data.

TPDU-NR and EOT 8 bits

Bit 7 (the high-order bit, bit mask 1000 0000) indicates the end of a XSDU (expedited TSDU). All other bits should be ZERO on output and ignored on input.

Note that the TP specification limits the size of an expedited transport service data unit (XSDU) to 16 octets.

RFC-1006 ISO Transport on top of the TCP

Comments

Since the release of RFC983 in April of 1986, we have gained much experience in using ISO transport services on top of the TCP. In September of 1986, we introduced the use of version 2 of the protocol, based mostly on comments from the community.

In January of 1987, we observed that the differences between version 2 of the protocol and the actual transport class 0 definition were actually quite small. In retrospect, this realization took much longerr than it should have: TP0 is is meant to run over a reliable network service, e.g., X.25. The TCP can be used to prrvide a service of this type, and, if no one complains to loudly, one could state that this memo really just describes a method for encapsulating TPO inside of TCP!

The changes in going from version 1 of the protocol to version 2 and then to version 3 are all relatively small. Initially, in describing version 1, we decided to use the TPDU formats from the ISO transport protocol. This naturally led to the evolution described above.

[GOSIP86]

The U.S. Government OSI User's Committee. "Government Open Systems Interconnection Procurement (GOSIP) Specification for Fiscal years 1987 and 1988." (December, 1986) [draft status]

[ISO8072] ISO

"International Standard 8072. Information Processing Systems -- Open Systems Interconnection: Transport Service Definition." (June, 1984)

[ISO8073] ISO

"International Standard 8073. Information Processing Systems -- Open Systems Interconnection: Transport Protocol Specification." (June, 1984)

[ISO8327] ISO

"International Standard 8327. Information Processing Systems -- Open Systems Interconnection: Session Protocol Specification." (June, 1984)

[X.214] CCITT.

"Recommendation X.214. Transport Service Definitions for Open Systems Interconnection (OSI) for CCITT Applications." (October, 1984)

[X.224] CCITT.

"Recommendation X.224. Transport Protocol Specification for Open Systems Interconnection (OSI) for CCITT Applications." (October, 1984)

[X.225] CCITT.

"Recommendation X.225. Session Protocol Specification for Open Systems Interconnection (OSI) for CCITT Applications." (October, 1984)

RFC-1014 XDR: External Data Representation Standard

Network Working Group
Sun Microsystems, Inc.
June 1987

This RFC describes a standard that Sun Microsystems, Inc., and others are using, one we wish to propose for the Internet's consideration. Distribution of this memo is unlimited.

RFC-1014 XDR: External Data Representation Standard

Introduction

XDR is a standard for the description and encoding of data. It is useful for transferring data between different computer architectures, and has been used to communicate data between such diverse machines as the SUN WORKSTATION*, VAX*, IBM-PC*, and Cray*. XDR fits into the ISO presentation layer, and is roughly analogous in purpose to X.409, ISO Abstract Syntax Notation. The major difference between these two is that XDR uses implicit typing, while X.409 uses explicit typing.

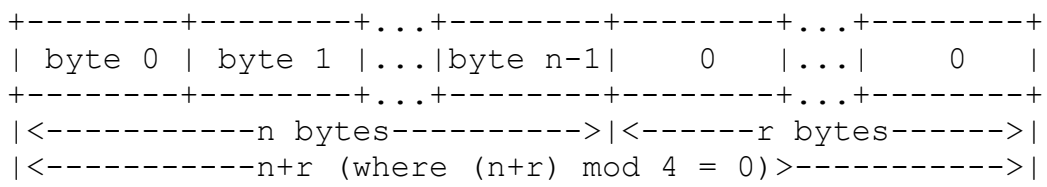
XDR uses a language to describe data formats. The language can be used only to describe data; it is not a programming language. This language allows one to describe intricate data formats in a concise manner. The alternative of using graphical representations (itself an informal language) quickly becomes incomprehensible when faced with complexity. The XDR language itself is similar to the C language, just as Courier is similar to Mesa. Protocols such as Sun RPC (Remote Procedure Call) and the NFS* (Network File System) use XDR to describe the format of their data.

The XDR standard makes the following assumption: that bytes (or octets) are portable, where a byte is defined to be 8 bits of data. A given hardware device should encode the bytes onto the various media in such a way that other hardware devices may decode the bytes without loss of meaning. For example, the Ethernet* standard suggests that bytes be encoded in "little-endian" style, or least significant bit first.

Basic Block Size

The representation of all items requires a multiple of four bytes (or 32 bits) of data. The bytes are numbered 0 through n-1. The bytes are read or written to some byte stream such that byte m always precedes byte m+1. If the n bytes needed to contain the data are not a multiple of four, then the n bytes are followed by enough (0 to 3) residual zero bytes, r, to make the total byte count a multiple of 4.

We include the familiar graphic box notation for illustration and comparison. In most illustrations, each box (delimited by a plus sign at the 4 corners and vertical bars and dashes) depicts a byte. Ellipses (...) between boxes show zero or more additional bytes where required.



BLOCK

XDR Data Types

Each of the sections that follow describes a data type defined in the XDR standard, shows how it is declared in the language, and includes a graphic illustration of its encoding.

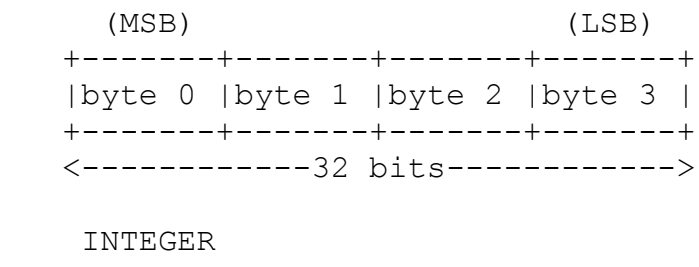
For each data type in the language we show a general paradigm declaration. Note that angle brackets (< and >) denote variablelength sequences of data and square brackets ([and]) denote fixed-length sequences of data. "n", "m" and "r" denote integers. For the full language specification and more formal definitions of terms such as "identifier" and "declaration", refer to the section: "The XDR Language Specification".

For some data types, more specific examples are included. A more extensive example of a data description is in the section: "An Example of an XDR Data Description".

Integer

An XDR signed integer is a 32-bit datum that encodes an integer in the range [-2147483648,2147483647]. The integer is represented in two's complement notation. The most and least significant bytes are 0 and 3, respectively. Integers are declared as follows:

int identifier;

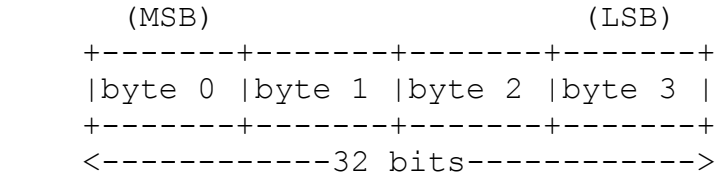


INTEGER

Unsigned Integer

An XDR unsigned integer is a 32-bit datum that encodes a nonnegative integer in the range [0,4294967295]. It is represented by an unsigned binary number whose most and least significant bytes are 0 and 3, respectively. An unsigned integer is declared as follows:

unsigned int identifier;



UNSIGNED INTEGER

Enumeration

Enumerations have the same representation as signed integers. Enumerations are handy for describing subsets of the integers. Enumerated data is declared as follows:

```
enum { name-identifier = constant, ... } identifier;
```

For example, the three colors red, yellow, and blue could be described by an enumerated type:

```
enum { RED = 2, YELLOW = 3, BLUE = 5 } colors;
```

It is an error to encode as an enum any other integer than those that have been given assignments in the enum declaration.

Boolean

Booleans are important enough and occur frequently enough to warrant their own explicit type in the standard. Booleans are declared as follows:

```
bool identifier;
```

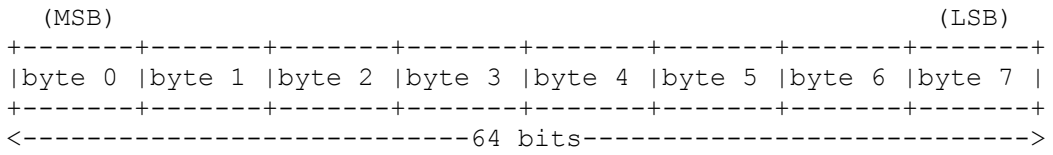
This is equivalent to:

```
enum { FALSE = 0, TRUE = 1 } identifier;
```

Hyper Integer and Unsigned Hyper Integer

The standard also defines 64-bit (8-byte) numbers called hyper integer and unsigned hyper integer. Their representations are the obvious extensions of integer and unsigned integer defined above. They are represented in two's complement notation. The most and least significant bytes are 0 and 7, respectively. Their declarations:

```
hyper identifier; unsigned hyper identifier;
```




```
      HYPER INTEGER
      UNSIGNED HYPER INTEGER
```

Floating-point

The standard defines the floating-point data type "float" (32 bits or 4 bytes). The encoding used is the IEEE standard for normalized single-precision floating-point numbers. The following three fields describe the single-precision floating-point number:

S: The sign of the number. Values 0 and 1 represent positive and negative, respectively. One bit.

E: The exponent of the number, base 2. 8 bits are devoted to this field. The exponent is biased by 127.

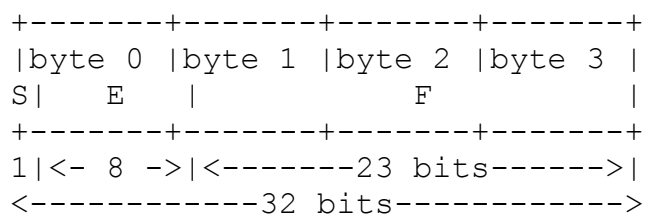
F: The fractional part of the number's mantissa, base 2. 23 bits are devoted to this field.

Therefore, the floating-point number is described by:

(-1)**S * 2**(E-Bias) * 1.F

It is declared as follows:

float identifier;



SINGLE-PRECISION FLOATING-POINT NUMBER

Just as the most and least significant bytes of a number are 0 and 3, the most and least significant bits of a single-precision floating- point number are 0 and 31. The beginning bit (and most significant bit) offsets of S, E, and F are 0, 1, and 9, respectively. Note that these numbers refer to the mathematical positions of the bits, and NOT to their actual physical locations (which vary from medium to medium).

The EEE specifications should be consulted concerning the encoding for signed zero, signed infinity (overflow), and denormalized numbers (underflow). According to IEEE specifications, the "NaN" (not a number) is system dependent and should not be used externally.

Double-precision Floating-point

The standard defines the encoding for the double-precision floating- point data type "double" (64 bits or 8 bytes). The encoding used is the IEEE standard for normalized double-precision floating-point numbers. The standard encodes the following three fields, which describe the double-precision floating-point number:

S: The sign of the number. Values 0 and 1 represent positive and negative, respectively. One bit.

E: The exponent of the number, base 2. 11 bits are devoted to this field. The exponent is biased by 1023.

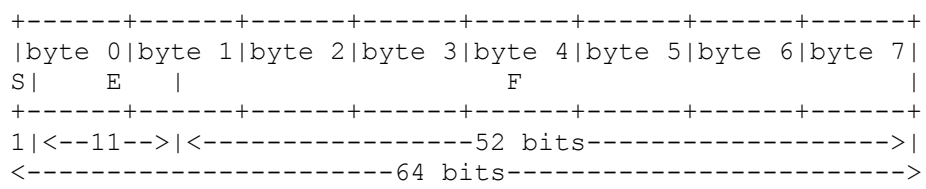
F: The fractional part of the number's mantissa, base 2. 52 bits are devoted to this field.

Therefore, the floating-point number is described by:

(-1)**S * 2**(E-Bias) * 1.F

It is declared as follows:

double identifier;



Just as the most and least significant bytes of a number are 0 and 3, the most and least significant bits of a double-precision floating- point number are 0 and 63. The beginning bit (and most significant bit) offsets of S, E , and F are 0, 1, and 12, respectively. Note that these numbers refer to the mathematical positions of the bits, and NOT to their actual physical locations (which vary from medium to medium).

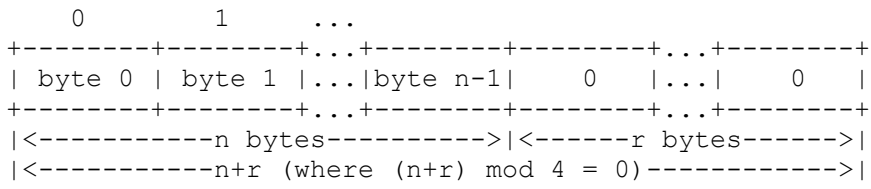
The IEEE specifications should be consulted concerning the encoding for signed zero, signed infinity (overflow), and denormalized numbers (underflow). According to IEEE specifications, the "NaN" (not a number) is system dependent and should not be used externally.

Fixed-length Opaque Data

At times, fixed-length uninterpreted data needs to be passed among machines. This data is called "opaque" and is declared as follows:

opaque identifier[n];

where the constant n is the (static) number of bytes necessary to contain the opaque data. If n is not a multiple of four, then the n bytes are followed by enough (0 to 3) residual zero bytes, r, to make the total byte count of the opaque object a multiple of four.



FIXED-LENGTH OPAQUE

Variable-length Opaque Data

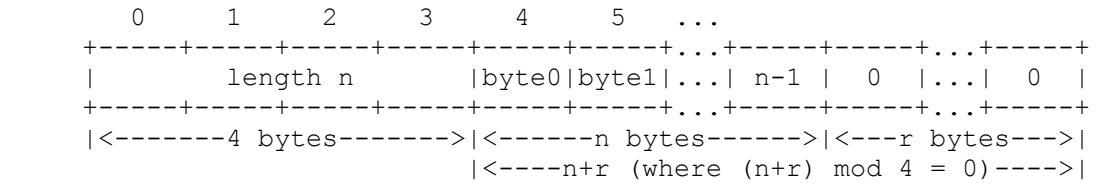
The standard also provides for variable-length (counted) opaque data, defined as a sequence of n (numbered 0 through n-1) arbitrary bytes to be the number n encoded as an unsigned integer (as described below), and followed by the n bytes of the sequence.

Byte m of the sequence always precedes byte m+1 of the sequence, and byte 0 of the sequence always follows the sequence's length (count). If n is not a multiple of four, then the n bytes are followed by enough (0 to 3) residual zero bytes, r, to make the total byte count a multiple of four. Variable-length opaque data is declared in the following way:

```
        opaque identifier<m>;
or
        opaque identifier<>;
```

The constant m denotes an upper bound of the number of bytes that the sequence may contain. If m is not specified, as in the second declaration, it is assumed to be (2**32) - 1, the maximum length. The constant m would normally be found in a protocol specification. For example, a filing protocol may state that the maximum data transfer size is 8192 bytes, as follows:

```
        opaque filedata<8192>;
```



VARIABLE-LENGTH OPAQUE

It is an error to encode a length greater than the maximum described in the specification.

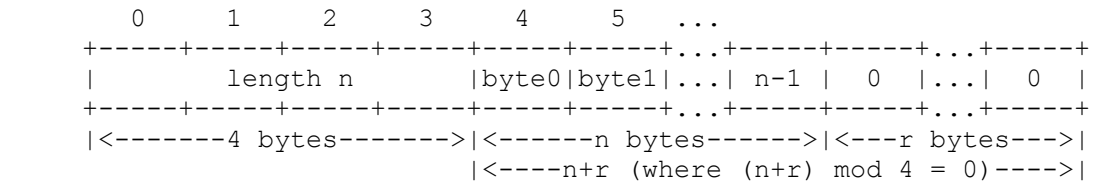
String

The standard defines a string of n (numbered 0 through n-1) ASCII bytes to be the number n encoded as an unsigned integer (as described above), and followed by the n bytes of the string. Byte m of the string always precedes byte m+1 of the string, and byte 0 of the string always follows the string's length. If n is not a multiple of four, then the n bytes are followed by enough (0 to 3) residual zero bytes, r, to make the total byte count a multiple of four. Counted byte strings are declared as follows:

```
   string object<m>;
or
   string object<>;
```

The constant m denotes an upper bound of the number of bytes that a string may contain. If m is not specified, as in the second declaration, it is assumed to be (2**32) - 1, the maximum length. The constant m would normally be found in a protocol specification. For example, a filing protocol may state that a file name can be no longer than 255 bytes, as follows:

```
   string filename<255>;
```



STRING

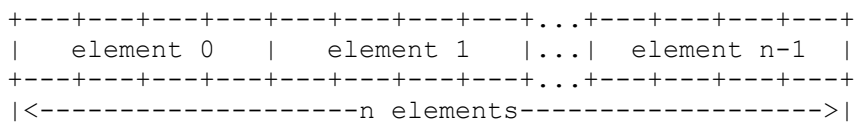
It is an error to encode a length greater than the maximum described in the specification.

Fixed-length Array

Declarations for fixed-length arrays of homogeneous elements are in the following form:

type-name identifier[n];

Fixed-length arrays of elements numbered 0 through n-1 are encoded by individually encoding the elements of the array in their natural order, 0 through n-1. Each element's size is a multiple of four bytes. Though all elements are of the same type, the elements may have different sizes. For example, in a fixed-length array of strings, all elements are of type "string", yet each element will vary in its length.



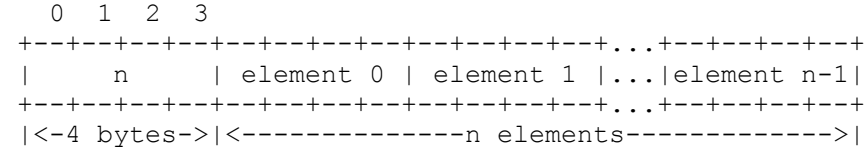
FIXED-LENGTH ARRAY

Variable-length Array

Counted arrays provide the ability to encode variable-length arrays of homogeneous elements. The array is encoded as the element count n (an unsigned integer) followed by the encoding of each of the array's elements, starting with element 0 and progressing through element n- 1. The declaration for variable-length arrays follows this form:

```
   type-name identifier<m>;
or
   type-name identifier<>;
```

The constant m specifies the maximum acceptable element count of an array; if m is not specified, as in the second declaration, it is assumed to be (2**32) - 1.



COUNTED ARRAY

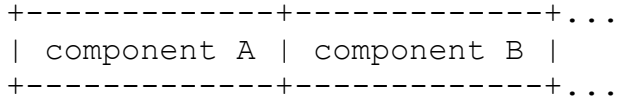
It is an error to encode a value of n that is greater than the maximum described in the specification.

Structure

Structures are declared as follows:

```
struct {
    component-declaration-A;
    component-declaration-B;
    ...
} identifier;
```

The components of the structure are encoded in the order of their declaration in the structure. Each component's size is a multiple of four bytes, though the components may be different sizes.



Structure

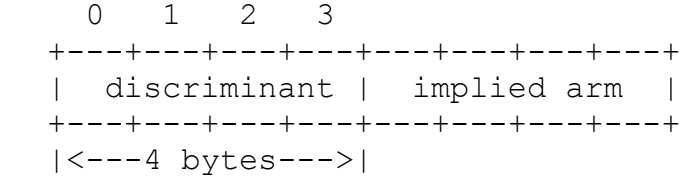
Discriminated Union

A discriminated union is a type composed of a discriminant followed by a type selected from a set of prearranged types according to the value of the discriminant. The type of discriminant is either "int", "unsigned int", or an enumerated type, such as "bool". The component types are called "arms" of the union, and are preceded by the value of the discriminant which implies their encoding. Discriminated unions are declared as follows:

```
union switch (discriminant-declaration) {
 case discriminant-value-A:
    arm-declaration-A;
 case discriminant-value-B:
    arm-declaration-B;
 ...
 default: default-declaration;
 } identifier;
```

Each "case" keyword is followed by a legal value of the discriminant. The default arm is optional. If it is not specified, then a valid encoding of the union cannot take on unspecified discriminant values. The size of the implied arm is always a multiple of four bytes.

The discriminated union is encoded as its discriminant followed by the encoding of the implied arm.



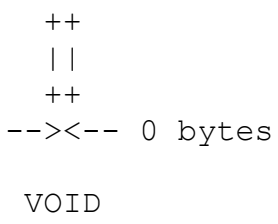
DISCRIMINATED UNION

Void

An XDR void is a 0-byte quantity. Voids are useful for describing operations that take no data as input or no data as output. They are also useful in unions, where some arms may contain data and others do not. The declaration is simply as follows:

```
void;
```

Voids are illustrated as follows:



Constant

The data declaration for a constant follows this form: "const" is used to define a symbolic name for a constant; it does not declare any data. The symbolic constant may be used anywhere a regular constant may be used. For example, the following defines a symbolic constant DOZEN, equal to 12.

```
const DOZEN = 12;
```

Typedef

"typedef" does not declare any data either, but serves to define new identifiers for declaring data. The syntax is:

```
typedef declaration;
```

The new type name is actually the variable name in the declaration part of the typedef. For example, the following defines a new type called "eggbox" using an existing type called "egg":

```
typedef egg eggbox[DOZEN];
```

Variables declared using the new type name have the same type as the new type name would have in the typedef, if it was considered a variable. For example, the following two declarations are equivalent in declaring the variable "fresheggs":

```
eggbox  fresheggs;
egg     fresheggs[DOZEN];
```

When a typedef involves a struct, enum, or union definition, there is another (preferred) syntax that may be used to define the same type. In general, a typedef of the following form:

```
typedef <<struct, union, or enum definition>> identifier;
```

may be converted to the alternative form by removing the "typedef" part and placing the identifier after the "struct", "union", or "enum" keyword, instead of at the end. For example, here are the two ways to define the type "bool":

```
typedef enum {     /* using typedef */
   FALSE = 0,
   TRUE = 1
} bool;

enum bool {        /* preferred alternative */
   FALSE = 0,
   TRUE = 1
};
```

The reason this syntax is preferred is one does not have to wait until the end of a declaration to figure out the name of the new type.

Optional-data

Optional-data is one kind of union that occurs so frequently that we give it a special syntax of its own for declaring it. It is declared as follows:

```
type-name *identifier;
```

This is equivalent to the following union:

```
union switch (bool opted) {
case TRUE:
   type-name element;
case FALSE:
   void;
} identifier;
```

It is also equivalent to the following variable-length array declaration, since the boolean "opted" can be interpreted as the length of the array:

```
type-name identifier<1>;
```

Optional-data is not so interesting in itself, but it is very useful for describing recursive data-structures such as linked-lists and trees. For example, the following defines a type "stringlist" that encodes lists of arbitrary length strings:

```
struct *stringlist {
   string item<>;
   stringlist next;
};
```

It could have been equivalently declared as the following union:

```
union stringlist switch (bool opted) {
case TRUE:
   struct {
      string item<>;
      stringlist next;
   } element;
case FALSE:
   void;
};
```

or as a variable-length array:

```
struct stringlist<1> {
   string item<>;
   stringlist next;
};
```

Both of these declarations obscure the intention of the stringlist type, so the optional-data declaration is preferred over both of them. The optional-data type also has a close correlation to how recursive data structures are represented in high-level languages such as Pascal or C by use of pointers. In fact, the syntax is the same as that of the C language for

pointers.

Areas for Future Enhancement

The XDR standard lacks representations for bit fields and bitmaps, since the standard is based on bytes. Also missing are packed (or binary-coded) decimals.

The intent of the XDR standard was not to describe every kind of data that people have ever sent or will ever want to send from machine to machine. Rather, it only describes the most commonly used data-types of high-level languages such as Pascal or C so that applications written in these languages will be able to communicate easily over some medium.

One could imagine extensions to XDR that would let it describe almost any existing protocol, such as TCP. The minimum necessary for this are support for different block sizes and byte-orders. The XDR discussed here could then be considered the 4-byte big-endian member of a larger XDR family.

RFC-1014 XDR: External Data Representation Standard

Discussion

1) Why use a language for describing data? What's wrong with diagrams?

There are many advantages in using a data-description language such as XDR versus using diagrams. Languages are more formal than diagrams and lead to less ambiguous descriptions of data. Languages are also easier to understand and allow one to think of other issues instead of the low-level details of bit-encoding. Also, there is a close analogy between the types of XDR and a high-level language such as C or Pascal. This makes the implementation of XDR encoding and decoding modules an easier task. Finally, the language specification itself is an ASCII string that can be passed from machine to machine to perform on-the-fly data interpretation.

2) Why is there only one byte-order for an XDR unit?

Supporting two byte-orderings requires a higher level protocol for determining in which byte-order the data is encoded. Since XDR is not a protocol, this can't be done. The advantage of this, though, is that data in XDR format can be written to a magnetic tape, for example, and any machine will be able to interpret it, since no higher level protocol is necessary for determining the byte-order.

3) Why is the XDR byte-order big-endian instead of little-endian? Isn't this unfair to little-endian machines such as the VAX(r), which has to convert from one form to the other?

Yes, it is unfair, but having only one byte-order means you have to be unfair to somebody. Many architectures, such as the Motorola 68000* and IBM 370*, support the big-endian byte-order.

4) Why is the XDR unit four bytes wide?

There is a tradeoff in choosing the XDR unit size. Choosing a small size such as two makes the encoded data small, but causes alignment problems for machines that aren't aligned on these boundaries. A large size such as eight means the data will be aligned on virtually every machine, but causes the encoded data to grow too big. We chose four as a compromise. Four is big enough to support most architectures efficiently, except for rare machines such as the eight-byte aligned Cray*. Four is also small enough to keep the encoded data restricted to a reasonable size.

5) Why must variable-length data be padded with zeros?

It is desirable that the same data encode into the same thing on all machines, so that encoded data can be meaningfully compared or checksummed. Forcing the padded bytes to be zero ensures this.

6) Why is there no explicit data-typing?

Data-typing has a relatively high cost for what small advantages it may have. One cost is the expansion of data due to the inserted type fields. Another is the added cost of interpreting these type fields and acting accordingly. And most protocols already know what type they expect, so data-typing supplies only redundant information. However, one can still get the benefits of data-typing using XDR. One way is to encode two things: first a string which is the XDR data description of the encoded data, and then the encoded data

itself. Another way is to assign a value to all the types in XDR, and then define a universal type which takes this value as its discriminant and for each value, describes the corresponding data type.

The XDR Language Specification

Notational Conventions

This specification uses an extended Back-Naur Form notation for describing the XDR language. Here is a brief description of the notation:

1) The characters '|', '(', ')', '[', ']', '"', and '*' are special.

2) Terminal symbols are strings of any characters surrounded by double quotes.

3) Non-terminal symbols are strings of non-special characters.

4) Alternative items are separated by a vertical bar ("|").

5) Optional items are enclosed in brackets.

6) Items are grouped together by enclosing them in parentheses.

7) A '*' following an item means 0 or more occurrences of that item.

For example, consider the following pattern:

```
"a " "very" (", " "very")* [" cold " "and "]  " rainy "
("day" | "night")
```

An infinite number of strings match this pattern. A few of them are:

```
"a very rainy day"
"a very, very rainy day"
"a very cold and  rainy day"
"a very, very, very cold and  rainy night"
```

Lexical Notes

1) Comments begin with '/*' and terminate with '*/'.

2) White space serves to separate items and is otherwise ignored.

3) An identifier is a letter followed by an optional sequence of letters, digits or underbar ('_'). The case of identifiers is not ignored.

4) A constant is a sequence of one or more decimal digits, optionally preceded by a minus-sign ('-').

Syntax Information

declaration:

```
     type-specifier identifier
   | type-specifier identifier "[" value "]"
   | type-specifier identifier "<" [ value ] ">"
   | "opaque" identifier "[" value "]"
   | "opaque" identifier "<" [ value ] ">"
   | "string" identifier "<" [ value ] ">"
   | type-specifier "*" identifier
   | "void"
```

value:

```
     constant
   | identifier
```

type-specifier:

```
     [ "unsigned" ] "int"
   | [ "unsigned" ] "hyper"
   | "float"
   | "double"
   | "bool"
   | enum-type-spec
   | struct-type-spec
   | union-type-spec
   | identifier
```

enum-type-spec:

```
   "enum" enum-body
```

enum-body:

```
   "{"
      ( identifier "=" value )
      ( "," identifier "=" value )*
   "}"
```

struct-type-spec:

```
   "struct" struct-body
```

struct-body:

```
   "{"
      ( declaration ";" )
      ( declaration ";" )*
   "}"
```

union-type-spec:

```
   "union" union-body
```

union-body:

```
   "switch" "(" declaration ")" "{"
      ( "case" value ":" declaration ";" )
      ( "case" value ":" declaration ";" )*
      [ "default" ":" declaration ";" ]
```

```
          "}"
```

constant-def:

```
          "const" identifier "=" constant ";"
```

type-def:

```
            "typedef" declaration ";"
          | "enum" identifier enum-body ";"
          | "struct" identifier struct-body ";"
          | "union" identifier union-body ";"
```

definition:

```
            type-def
          | constant-def
```

specification:

```
            definition *
```

Syntax Notes

1) The following are keywords and cannot be used as identifiers: "bool", "case", "const", "default", "double", "enum", "float", "hyper", "opaque", "string", "struct", "switch", "typedef", "union", "unsigned" and "void".

2) Only unsigned constants may be used as size specifications for arrays. If an identifier is used, it must have been declared previously as an unsigned constant in a "const" definition.

3) Constant and type identifiers within the scope of a specification are in the same name space and must be declared uniquely within this scope.

4) Similarly, variable names must be unique within the scope of struct and union declarations. Nested struct and union declarations create new scopes.

5) The discriminant of a union must be of a type that evaluates to an integer. That is, "int", "unsigned int", "bool", an enumerated type or any typedefed type that evaluates to one of these is legal. Also, the case values must be one of the legal values of the discriminant. Finally, a case value may not be specified more than once within the scope of a union declaration.

An Example of an XDR Data Description

Here is a short XDR data description of a thing called a "file", which might be used to transfer files from one machine to another.

```
const MAXUSERNAME = 32;     /* max length of a user name */
const MAXFILELEN = 65535;   /* max length of a file      */
const MAXNAMELEN = 255;     /* max length of a file name */

/*
 * Types of files:
 */
enum filekind {
    TEXT = 0,       /* ascii data */
    DATA = 1,       /* raw data   */
    EXEC = 2        /* executable */
};

/*
 * File information, per kind of file:
 */
union filetype switch (filekind kind) {
case TEXT:
    void;                                /* no extra information */
case DATA:
    string creator<MAXNAMELEN>;      /* data creator           */
case EXEC:
    string interpretor<MAXNAMELEN>; /* program interpretor   */
};

/*
 * A complete file:
 */
struct file {
    string filename<MAXNAMELEN>; /* name of file     */
    filetype type;                  /* info about file */
    string owner<MAXUSERNAME>;   /* owner of file    */
    opaque data<MAXFILELEN>;     /* file data        */
};
```

Suppose now that there is a user named "john" who wants to store his lisp program "sillyprog" that contains just the data "(quit)". His file would be encoded as follows:

```
OFFSET  HEX BYTES       ASCII    COMMENTS
------  ---------       -----    --------
 0      00 00 00 09     ....     -- length of filename = 9
 4      73 69 6c 6c     sill     -- filename characters
 8      79 70 72 6f     ypro     -- ... and more characters ...
12      67 00 00 00     g...     -- ... and 3 zero-bytes of fill
16      00 00 00 02     ....     -- filekind is EXEC = 2
20      00 00 00 04     ....     -- length of interpretor = 4
24      6c 69 73 70     lisp     -- interpretor characters
28      00 00 00 04     ....     -- length of owner = 4
```

```
32      6a 6f 68 6e     john    -- owner characters
36      00 00 00 06     ....    -- length of file data = 6
40      28 71 75 69     (qui    -- file data bytes ...
44      74 29 00 00     t)..    -- ... and 2 zero-bytes of fill
```

Brian W. Kernighan & Dennis M. Ritchie, "The C Programming Language", Bell Laboratories, Murray Hill, New Jersey, 1978.

Danny Cohen, "On Holy Wars and a Plea for Peace", IEEE Computer, October 1981.

"IEEE Standard for Binary Floating-Point Arithmetic", ANSI/IEEE Standard 754-1985, Institute of Electrical and Electronics Engineers, August 1985.

"Courier: The Remote Procedure Call Protocol", XEROX Corporation, XSIS 038112, December 1981.

Trademarks and Owners

SUN WORKSTATION	Sun Microsystems, Inc.
VAX	Digital EquipmentCorporation
IBM PC	International Business Machines Corporation
Cray	Cray Research
NFS	Sun Microsystems, Inc.
Ethernet	Xerox Corporation.
Motorola 68000	Motorola, Inc.
IBM 370	International Business Machines Corporation

RFC-1032 Domain Administrators Guide

M. Stahl
SRI International
November 1987

This memo describes procedures for registering a domain with the Network Information Center (NIC) of Defense Data Network (DDN), and offers guidelines on the establishment and administration of a domain in accordance with the requirements specified in RFC-920. It is intended for use by domain administrators. This memo should be used in conjunction with RFC-920, which is an official policy statement of the Internet Activities Board (IAB) and the Defense Advanced Research Projects Agency (DARPA).

RFC-1032 Domain Administrators Guide

Background

Domains are administrative entities that provide decentralized management of host naming and addressing. The domain-naming system is distributed and hierarchical.

The NIC is designated by the Defense Communications Agency (DCA) to provide registry services for the domain-naming system on the DDN and DARPA portions of the Internet.

As registrar of top-level and second-level domains, as well as administrator of the root domain name servers on behalf of DARPA and DDN, the NIC is responsible for maintaining the root server zone files and their binary equivalents. In addition, the NIC is responsible for administering the top-level domains of "ARPA," "COM," "EDU," "ORG," "GOV," and "MIL" on behalf of DCA and DARPA until it becomes feasible for other appropriate organizations to assume those responsibilities.

It is recommended that the guidelines described in this document be used by domain administrators in the establishment and control of second-level domains.

RFC-1032 Domain Administrators Guide

The Domain Administrator

The role of the domain administrator (DA) is that of coordinator, manager, and technician. If his domain is established at the second level or lower in the tree, the DA must register by interacting with the management of the domain directly above his, making certain that his domain satisfies all the requirements of the administration under which his domain would be situated. To find out who has authority over the name space he wishes to join, the DA can ask the NIC Hostmaster. Information on contacts for the top-level and second- level domains can also be found on line in the file `NETINFO:DOMAIN- CONTACTS.TXT`, which is available from the NIC via anonymous FTP.

The DA should be technically competent; he should understand the concepts and procedures for operating a domain server, as described in RFC-1034, and make sure that the service provided is reliable and uninterrupted. It is his responsibility or that of his delegate to ensure that the data will be current at all times. As a manager, the DA must be able to handle complaints about service provided by his domain name server. He must be aware of the behavior of the hosts in his domain, and take prompt action on reports of problems, such as protocol violations or other serious misbehavior. The administrator of a domain must be a responsible person who has the authority to either enforce these actions himself or delegate them to someone else.

Name assignments within a domain are controlled by the DA, who should verify that names are unique within his domain and that they conform to standard naming conventions. He furnishes access to names and name-related information to users both inside and outside his domain. He should work closely with the personnel he has designated as the "technical and zone" contacts for his domain, for many administrative decisions will be made on the basis of input from these people.

The Domain Technical and Zone Contact

A zone consists of those contiguous parts of the domain tree for which a domain server has complete information and over which it has authority. A domain server may be authoritative for more than one zone. The domain technical/zone contact is the person who tends to the technical aspects of maintaining the domain's name server and resolver software, and database files. He keeps the name server running, and interacts with technical people in other domains and zones to solve problems that affect his zone.

Policies

Domain or host name choices and the allocation of domain name space are considered to be local matters. In the event of conflicts, it is the policy of the NIC not to get involved in local disputes or in the local decision-making process. The NIC will not act as referee in disputes over such matters as who has the "right" to register a particular top-level or second-level domain for an organization. The NIC considers this a private local matter that must be settled among the parties involved prior to their commencing the registration process with the NIC. Therefore, it is assumed that the responsible person for a domain will have resolved any local conflicts among the members of his domain before registering that domain with the NIC. The NIC will give guidance, if requested, by answering specific technical questions, but will not provide arbitration in disputes at the local level. This policy is also in keeping with the distributed hierarchical nature of the domain-naming system in that it helps to distribute the tasks of solving problems and handling questions.

Naming conventions for hosts should follow the rules specified in RFC-952. From a technical standpoint, domain names can be very long. Each segment of a domain name may contain up to 64 characters, but the NIC strongly advises DAs to choose names that are 12 characters or fewer, because behind every domain system there is a human being who must keep track of the names, addresses, contacts, and other data in a database. The longer the name, the more likely the data maintainer is to make a mistake. Users also will appreciate shorter names. Most people agree that short names are easier to remember and type; most domain names registered so far are 12 characters or fewer.

Domain name assignments are made on a first-come-first-served basis. The NIC has chosen not to register individual hosts directly under the top-level domains it administers. One advantage of the domain naming system is that administration and data maintenance can be delegated down a hierarchical tree. Registration of hosts at the same level in the tree as a second-level domain would dilute the usefulness of this feature. In addition, the administrator of a domain is responsible for the actions of hosts within his domain. We would not want to find ourselves in the awkward position of policing the actions of individual hosts. Rather, the subdomains registered under these top-level domains retain the responsibility for this function.

Countries that wish to be registered as top-level domains are required to name themselves after the two-letter country code listed in the international standard ISO-3166. In some cases, however, the two-letter ISO country code is identical to a state code used by the U.S. Postal Service. Requests made by countries to use the three- letter form of country code specified in the ISO-3166 standard will be considered in such cases so as to prevent possible conflicts and confusion.

How To Register

Obtain a domain questionnaire from the NIC hostmaster, or FTP the file `NETINFO:DOMAIN-TEMPLATE.TXT` from host `SRI-NIC.ARPA`.

Fill out the questionnaire completely. Return it via electronic mail to `HOSTMASTER@SRI-NIC.ARPA`.

The APPENDIX to this memo contains the application form for registering a top-level or second-level domain with the NIC. It supersedes the version of the questionnaire found in RFC-920. The application should be submitted by the person administratively responsible for the domain, and must be filled out completely before the NIC will authorize establishment of a top-level or second-level domain. The DA is responsible for keeping his domain's data current with the NIC or with the registration agent with which his domain is registered. For example, the CSNET and UUCP managements act as domain filters, processing domain applications for their own organizations. They pass pertinent information along periodically to the NIC for incorporation into the domain database and root server files. The online file `NETINFO:ALTERNATE-DOMAIN-PROCEDURE.TXT` outlines this procedure. It is highly recommended that the DA review this information periodically and provide any corrections or additions. Corrections should be submitted via electronic mail.

Which Domain Name?

The designers of the domain-naming system initiated several general categories of names as top-level domain names, so that each could accommodate a variety of organizations. The current top-level domains registered with the DDN Network Information Center are ARPA, COM, EDU, GOV, MIL, NET, and ORG, plus a number of top-level country domains. To join one of these, a DA needs to be aware of the purpose for which it was intended.

"ARPA" is a temporary domain. It is by default appended to the names of hosts that have not yet joined a domain. When the system was begun in 1984, the names of all hosts in the Official DoD Internet Host Table maintained by the NIC were changed by adding of the label ".ARPA" in order to accelerate a transition to the domain-naming system. Another reason for the blanket name changes was to force hosts to become accustomed to using the new style names and to modify their network software, if necessary. This was done on a network-wide basis and was directed by DCA in DDN Management Bulletin No. 22. Hosts that fall into this domain will eventually move to other branches of the domain tree.

> "COM" is meant to incorporate subdomains of companies and businesses.
>
> "EDU" was initiated to accommodate subdomains set up by universities and other educational institutions.
>
> "GOV" exists to act as parent domain for subdomains set up by government agencies.
>
> "MIL" was initiated to act as parent to subdomains that are developed by military organizations.
>
> "NET" was introduced as a parent domain for various network-type organizations. Organizations that belong within this top-level domain are generic or network-specific, such as network service centers and consortia. "NET" also encompasses network management-related organizations, such as information centers and operations centers.
>
> "ORG" exists as a parent to subdomains that do not clearly fall within the other top-level domains. This may include technical- support groups, professional societies, or similar organizations.

One of the guidelines in effect in the domain-naming system is that a host should have only one name regardless of what networks it is connected to. This implies, that, in general, domain names should not include routing information or addresses. For example, a host that has one network connection to the Internet and another to BITNET should use the same name when talking to either network. RFC-1034 contains a description of the syntax of domain names.

Verification Of Data

The verification process can be accomplished in several ways. One of these is through the NIC WHOIS server. If he has access to WHOIS, the DA can type the command "`whois domain <domain name><return>`". The reply from WHOIS will supply the following: the name and address of the organization "owning" the domain; the name of the domain; its administrative, technical, and zone contacts; the host names and network addresses of sites providing name service for the domain.

Example:

```
@whois domain rice.edu<Return>

Rice University (RICE-DOM)
Advanced Studies and Research
Houston, TX 77001

Domain Name: RICE.EDU

Administrative Contact:
Kennedy, Ken  (KK28)  Kennedy@LLL-CRG.ARPA (713) 527-4834
Technical Contact, Zone Contact:
Riffle, Vicky R.  (VRR)  rif@RICE.EDU
(713) 527-8101 ext 3844
```

Domain servers:

```
RICE.EDU                      128.42.5.1
PENDRAGON.CS.PURDUE.EDU       128.10.2.5
```

Alternatively, the DA can send an electronic mail message to `SERVICE@SRI-NIC.ARPA`. In the subject line of the message header, the DA should type "`whois domain <domain name>`". The requested information will be returned via electronic mail. This method is convenient for sites that do not have access to the NIC WHOIS service.

The initial application for domain authorization should be submitted via electronic mail, if possible, to `HOSTMASTER@SRI-NIC.ARPA`. The questionnaire described in the appendix may be used or a separate application can be FTPed from host `SRI-NIC.ARPA`. The information provided by the administrator will be reviewed by hostmaster personnel for completeness. There will most likely be a few exchanges of correspondence via electronic mail, the preferred method of communication, prior to authorization of the domain.

How To Get More Information

An informational table of the top-level domains and their root servers is contained in the file NETINFO:DOMAINS.TXT online at SRI- NIC.ARPA. This table can be obtained by FTPing the file. Alternatively, the information can be acquired by opening a TCP or UDP connection to the NIC Host Name Server, port 101 on SRI-NIC.ARPA, and invoking the command "ALL-DOM".

The following online files, all available by FTP from SRI-NIC.ARPA, contain pertinent domain information:

- NETINFO:DOMAINS.TXT, a table of all top-level domains and the network addresses of the machines providing domain name service for them. It is updated each time a new top-level domain is approved.
- NETINFO:DOMAIN-INFO.TXT contains a concise list of all top-level and second-level domain names registered with the NIC and is updated monthly.
- NETINFO:DOMAIN-CONTACTS.TXT also contains a list of all the top level and second-level domains, but includes the administrative, technical and zone contacts for each as well.
- NETINFO:DOMAIN-TEMPLATE.TXT contains the questionnaire to be completed before registering a top-level or second-level domain.

For either general or specific information on the domain system, do one or more of the following:

- Send electronic mail to HOSTMASTER@SRI-NIC.ARPA
- Call the toll-free NIC hotline at (800) 235-3155

RFC-1032 Domain Administrators Guide: Appendix

Registration Form

The following questionnaire may be FTPed from SRI-NIC.ARPA as NETINFO:DOMAIN-TEMPLATE.TXT.

To establish a domain, the following information must be sent to the NIC Domain Registrar (HOSTMASTER@SRI-NIC.ARPA):

NOTE: The key people must have electronic mailboxes and NIC "handles," unique NIC database identifiers. If you have access to "WHOIS", please check to see if you are registered and if so, make sure the information is current. Include only your handle and any changes (if any) that need to be made in your entry. If you do not have access to "WHOIS", please provide all the information indicated and a NIC handle will be assigned.

(1) The name of the top-level domain to join.

For example: COM

(2) The NIC handle of the administrative head of the organization. Alternately, the person's name, title, mailing address, phone number, organization, and network mailbox. This is the contact point for administrative and policy questions about the domain. In the case of a research project, this should be the principal investigator.

For example:

Administrator

```
Organization  The NetWorthy Corporation
Name          Penelope Q. Sassafrass
Title         President
Mail Address  The NetWorthy Corporation
           4676 Andrews Way, Suite 100
           Santa Clara, CA 94302-1212
Phone Number  (415) 123-4567
Net Mailbox   Sassafrass@ECHO.TNC.COM
NIC Handle    PQS
```

(3) The NIC handle of the technical contact for the domain. Alternately, the person's name, title, mailing address, phone number, organization, and network mailbox. This is the contact point for problems concerning the domain or zone, as well as for updating information about the domain or zone.

For example:

Technical and Zone Contact

```
Organization  The NetWorthy Corporation
Name          Ansel A. Aardvark
Title         Executive Director
Mail Address  The NetWorthy Corporation
           4676 Andrews Way, Suite 100
```

```
                Santa Clara, CA. 94302-1212
Phone Number    (415) 123-6789
Net Mailbox     Aardvark@ECHO.TNC.COM
NIC Handle      AAA2
```

(4) The name of the domain (up to 12 characters). This is the name that will be used in tables and lists associating the domain with the domain server addresses. [While, from a technical standpoint, domain names can be quite long (programmers beware), shorter names are easier for people to cope with.]

For example: TNC

(5) A description of the servers that provide the domain service for translating names to addresses for hosts in this domain, and the date they will be operational.

For example: Our server is a copy of the one operated by the NIC; it will be installed and made operational on 1 November 1987.

(6) Domains must provide at least two independent servers for the domain. Establishing the servers in physically separate locations and on different PSNs is strongly recommended. A description of the server machine and its backup, including

(a) Hardware and software (using keywords from the Assigned Numbers RFC).

(b) Host domain name and network addresses (which host on which network for each connected network).

(c) Any domain-style nicknames (please limit your domain-style nickname request to one)

For example:

- Hardware and software

```
VAX-11/750  and  UNIX,    or
IBM-PC      and  MS-DOS,  or
DEC-1090    and  TOPS-20
```

- Host domain names and network addresses

`BAR.FOO.COM 10.9.0.193` on ARPANET

- Domain-style nickname

`BR.FOO.COM` (same as `BAR.FOO.COM 10.9.0.13` on ARPANET)

(7) Planned mapping of names of any other network hosts, other than the server machines, into the new domain's naming space.

For example:

```
BAR-FOO2.ARPA (10.8.0.193) -> FOO2.BAR.COM
BAR-FOO3.ARPA (10.7.0.193) -> FOO3.BAR.COM
```

BAR-FOO4.ARPA (10.6.0.193) -> FOO4.BAR.COM

(8) An estimate of the number of hosts that will be in the domain.

(a) Initially
(b) Within one year
(c) Two years
(d) Five years.

For example:

(a) Initially = 50
(b) One year = 100
(c) Two years = 200
(d) Five years = 500

(9) The date you expect the fully qualified domain name to become the official host name in HOSTS.TXT.

Please note: If changing to a fully qualified domain name (e.g., FOO.BAR.COM) causes a change in the official host name of an ARPANET or MILNET host, DCA approval must be obtained beforehand. Allow 10 working days for your requested changes to be processed.

ARPANET sites should contact ARPANETMGR@DDN1.ARPA. MILNET sites should contact HOSTMASTER@SRI-NIC.ARPA, 800-235-3155, for further instructions.

(10) Please describe your organization briefly.

For example: The NetWorthy Corporation is a consulting organization of people working with UNIX and the C language in an electronic networking environment. It sponsors two technical conferences annually and distributes a bimonthly newsletter.

RFC-1032 Domain Administrators Guide: Appendix

Example of Completed Registration Form

This example of a completed application corresponds to the examples found in the companion document (RFC-1033) entitled "Domain Administrators Operations Guide."

(1) The name of the top-level domain to join.

COM

(2) The NIC handle of the administrative contact person.

NIC Handle JAKE

(3) The NIC handle of the domain's technical and zone contact person.

NIC Handle DLE6

(4) The name of the domain.

SRI

(5) A description of the servers.

Our server is the TOPS20 server JEEVES supplied by ISI; it will be installed and made operational on 1 July 1987.

(6) A description of the server machine and its backup:

(a) Hardware and software

DEC-1090T and TOPS20
DEC-2065 and TOPS20

(b) Host domain name and network address

```
KL.SRI.COM  10.1.0.2 on ARPANET, 128.18.10.6 on SRINET
STRIPE.SRI.COM  10.4.0.2 on ARPANET, 128.18.10.4 on SRINET
```

(c) Domain-style nickname

None

(7) Planned mapping of names of any other network hosts, other than the server machines, into the new domain's naming space.

```
SRI-Blackjack.ARPA (128.18.2.1) -> Blackjack.SRI.COM
SRI-CSL.ARPA (192.12.33.2) -> CSL.SRI.COM
```

(8) An estimate of the number of hosts that will be directly within this domain.

(a) Initially = 50
(b) One year = 100
(c) Two years = 200

(d) Five years = 500

(9) A date when you expect the fully qualified domain name to become the official host name in `HOSTS.TXT`.

31 September 1987

(10) Brief description of organization.

SRI International is an independent, nonprofit, scientific research organization. It performs basic and applied research for government and commercial clients, and contributes to worldwide economic, scientific, industrial, and social progress through research and related services.

[ISO-3166] "Codes for the Representation of Names of Countries", ISO-3166, International Standards Organization, May 1981.

[Not online]

RFC-1033 Domain Administrators Operations Guide

M. Lottor
SRI International
November 1987

This RFC provides guidelines for domain administrators in operating a domain server and maintaining their portion of the hierarchical database. Familiarity with the domain system is assumed. Distribution of this memo is unlimited.

RFC-1033--Domain Administrators Operations Guide

Introduction

A domain server requires a few files to get started. It will normally have some number of boot/startup files (also known as the "safety belt" files). One section will contain a list of possible root servers that the server will use to find the up-to-date list of root servers. Another section will list the zone files to be loaded into the server for your local domain information. A zone file typically contains all the data for a particular domain. This guide describes the data formats that can be used in zone files and suggested parameters to use for certain fields. If you are attempting to do anything advanced or tricky, consult the appropriate domain RFC's for more details.

Note: Each implementation of domain software may require different files. Zone files are standardized but some servers may require other startup files. See the appropriate documentation that comes with your software. See the appendix for some specific examples.

Zones

A zone defines the contents of a contiguous section of the domain space, usually bounded by administrative boundaries. There will typically be a separate data file for each zone. The data contained in a zone file is composed of entries called Resource Records (RRs).

You may only put data in your domain server that you are authoritative for. You must not add entries for domains other than your own (except for the special case of "glue records").

A domain server will probably read a file on start-up that lists the zones it should load into its database. The format of this file is not standardized and is different for most domain server implementations. For each zone it will normally contain the domain name of the zone and the file name that contains the data to load for the zone.

Root Servers

A resolver will need to find the root servers when it first starts. When the resolver boots, it will typically read a list of possible root servers from a file.

The resolver will cycle through the list trying to contact each one. When it finds a root server, it will ask it for the current list of root servers. It will then discard the list of root servers it read from the data file and replace it with the current list it received.

Root servers will not change very often. You can get the names of current root servers from the NIC.

FTP the file `NETINFO:ROOT-SERVERS.TXT` or send a mail request to `NIC@SRI-NIC.ARPA`.

As of this date (June 1987) they are:

```
SRI-NIC.ARPA       10.0.0.51    26.0.0.73
C.ISI.EDU          10.0.0.52
BRL-AOS.ARPA       192.5.25.82  192.5.22.82   128.20.1.2
A.ISI.EDU          26.3.0.103
```

RFC-1033--Domain Administrators Operations Guide

Resource Records

Records in the zone data files are called resource records (RRs). They are specified in RFC-1034 and RFC-1035.

RFC-1033--Domain Operations Guide: Resource Records

Format

An RR has a standard format as shown:

```
<name>   [<ttl>]   [<class>]   <type>   <data>
```

The record is divided into fields which are separated by white space.

`<name>`

The name field defines what domain name applies to the given RR. In some cases the name field can be left blank and it will default to the name field of the previous RR.

`<ttl>`

TTL stands for Time To Live. It specifies how long a domain resolver should cache the RR before it throws it out and asks a domain server again. See the section on TTL's. If you leave the TTL field blank it will default to the minimum time specified in the SOA record (described later).

`<class>`

The class field specifies the protocol group. If left blank it will default to the last class specified.

`<type>`

The type field specifies what type of data is in the RR. See the section on types.

`<data>`

The data field is defined differently for each type and class of data. Popular RR data formats are described later.

The domain system does not guarantee to preserve the order of resource records. Listing RRs (such as multiple address records) in a certain order does not guarantee they will be used in that order.

Case is preserved in names and data fields when loaded into the name server. All comparisons and lookups in the name server are case insensitive.

Parenthesis ("(",")") are used to group data that crosses a line boundary.

A semicolon (";") starts a comment; the remainder of the line is ignored.

The asterisk ("*") is used for wildcarding.

The at-sign ("@") denotes the current default domain name.

Names

A domain name is a sequence of labels separated by dots.

Domain names in the zone files can be one of two types, either absolute or relative. An absolute name is the fully qualified domain name and is terminated with a period. A relative name does not terminate with a period, and the current default domain is appended to it. The default domain is usually the name of the domain that was specified in the boot file that loads each zone.

The domain system allows a label to contain any 8-bit character. Although the domain system has no restrictions, other protocols such as SMTP do have name restrictions. Because of other protocol restrictions, only the following characters are recommended for use in a host name (besides the dot separator):

"A-Z", "a-z", "0-9", dash and underscore

RFC-1033--Domain Operations Guide: Resource Records

TTL's (Time to Live)

It is important that TTLs are set to appropriate values. The TTL is the time (in seconds) that a resolver will use the data it got from your server before it asks your server again. If you set the value too low, your server will get loaded down with lots of repeat requests. If you set it too high, then information you change will not get distributed in a reasonable amount of time. If you leave the TTL field blank, it will default to what is specified in the SOA record for the zone.

Most host information does not change much over long time periods. A good way to set up your TTLs would be to set them at a high value, and then lower the value if you know a change will be coming soon. You might set most TTLs to anywhere between a day (86400) and a week (604800). Then, if you know some data will be changing in the near future, set the TTL for that RR down to a lower value (an hour to a day) until the change takes place, and then put it back up to its previous value.

Also, all RRs with the same name, class, and type should have the same TTL value.

Classes

The domain system was designed to be protocol independent. The class field is used to identify the protocol group that each RR is in.

The class of interest to people using TCP/IP software is the class "Internet". Its standard designation is "IN".

A zone file should only contain RRs of the same class.

Types

There are many defined RR types. For a complete list, see the domain specification RFCs. Here is a list of current commonly used types. The data for each type is described in the data section.

Designation	Description
SOA	Start Of Authority
NS	Name Server
A	Internet Address
CNAME	Canonical Name (nickname pointer)
HINFO	Host Information
WKS	Well Known Services
MX	Mail Exchanger
PTR	Pointer

RFC-1033--Domain Operations Guide: Resource Records

SOA (Start Of Authority)

```
<name>  [<ttl>]  [<class>]  SOA  <origin>  <person>  (
                <serial>
                <refresh>
                <retry>
                <expire>
                <minimum> )
```

The Start Of Authority record designates the start of a zone. The zone ends at the next SOA record.

<name> is the name of the zone.

<origin> is the name of the host on which the master zone file resides.

<person> is a mailbox for the person responsible for the zone. It is formatted like a mailing address but the at-sign that normally separates the user from the host name is replaced with a dot.

<serial> is the version number of the zone file. It should be incremented anytime a change is made to data in the zone.

<refresh> is how long, in seconds, a secondary name server is to check with the primary name server to see if an update is needed. A good value here would be one hour (3600).

<retry> is how long, in seconds, a secondary name server is to retry after a failure to check for a refresh. A good value here would be 10 minutes (600).

<expire> is the upper limit, in seconds, that a secondary name server is to use the data before it expires for lack of getting a refresh. You want this to be rather large, and a nice value is 3600000, about 42 days.

<minimum> is the minimum number of seconds to be used for TTL values in RRs. A minimum of at least a day is a good value here (86400).

There should only be one SOA record per zone. A sample SOA record would look something like:

```
@   IN   SOA   SRI-NIC.ARPA.   HOSTMASTER.SRI-NIC.ARPA. (
                45          ;serial
                3600        ;refresh
                600         ;retry
                3600000     ;expire
                86400 )     ;minimum
```

RFC-1033--Domain Operations Guide: Resource Records

NS (Name Server)

```
<domain>   [<ttl>] [<class>]   NS   <server>
```

The NS record lists the name of a machine that provides domain service for a particular domain. The name associated with the RR is the domain name and the data portion is the name of a host that provides the service. If machines `SRI-NIC.ARPA` and `C.ISI.EDU` provide name lookup service for the domain COM then the following entries would be used:

```
      COM.     NS       SRI-NIC.ARPA.
               NS       C.ISI.EDU.
```

Note that the machines providing name service do not have to live in the named domain. There should be one NS record for each server for a domain. Also note that the name "COM" defaults for the second NS record.

NS records for a domain exist in both the zone that delegates the domain, and in the domain itself.

Glue Records

If the name server host for a particular domain is itself inside the domain, then a 'glue' record will be needed. A glue record is an A (address) RR that specifies the address of the server. Glue records are only needed in the server delegating the domain, not in the domain itself. If for example the name server for domain SRI.COM was KL.SRI.COM, then the NS record would look like this, but you will also need to have the following A record.

```
SRI.COM.      NS      KL.SRI.COM.
KL.SRI.COM.   A       10.1.0.2
```

A (Address)

<host> [<ttl>] [<class>] A <address>

The data for an A record is an internet address in dotted decimal form. A sample A record might look like:

```
SRI-NIC.ARPA.           A       10.0.0.51
```

There should be one A record for each address of a host.

CNAME (Canonical Name)

<nickname> [<ttl>] [<class>] CNAME <host>

The CNAME record is used for nicknames. The name associated with the RR is the nickname. The data portion is the official name. For example, a machine named SRI-NIC.ARPA may want to have the nickname NIC.ARPA. In that case, the following RR would be used:

```
NIC.ARPA.       CNAME   SRI-NIC.ARPA.
```

There must not be any other RRs associated with a nickname of the same class.

Nicknames are also useful when a host changes it's name. In that case, it is usually a good idea to have a CNAME pointer so that people still using the old name will get to the right place.

RFC-1033--Domain Operations Guide: Resource Records

HINFO (Host Info)

<host> [<ttl>] [<class>] HINFO <hardware> <software>

The HINFO record gives information about a particular host. The data is two strings separated by whitespace. The first string is a hardware description and the second is software. The hardware is usually a manufacturer name followed by a dash and model designation. The software string is usually the name of the operating system.

Official HINFO types can be found in the latest **Assigned Numbers RFC**, the latest of which is **RFC-1060**. The Hardware type is called the Machine name and the Software type is called the System name.

Some sample HINFO records:

```
SRI-NIC.ARPA.           HINFO   DEC-2060 TOPS20
UCBARPA.Berkeley.EDU.   HINFO   VAX-11/780 UNIX
```

RFC-1033--Domain Operations Guide: Resource Records

WKS (Well Known Services)

<host> [<ttl>] [<class>] WKS <address> <protocol> <services>

The WKS record is used to list Well Known Services a host provides. WKS's are defined to be services on port numbers below 256. The WKS record lists what services are available at a certain address using a certain protocol. The common protocols are TCP or UDP. A sample WKS record for a host offering the same services on all address would look like:

Official protocol names can be found in the latest **Assigned Numbers RFC**, the latest of which is **RFC-1060**.

```
SRI-NIC.ARPA.   WKS  10.0.0.51  TCP  TELNET FTP SMTP
                WKS  10.0.0.51  UDP  TIME
                WKS  26.0.0.73  TCP  TELNET FTP SMTP
                WKS  26.0.0.73  UDP  TIME
```

RFC-1033--Domain Operations Guide: Resource Records

MX (Mail Exchanger) (See RFC-974 for more details.)

<name> [<ttl>] [<class>] MX <preference> <host>

MX records specify where mail for a domain name should be delivered. There may be multiple MX records for a particular name. The preference value specifies the order a mailer should try multiple MX records when delivering mail. Zero is the highest preference. Multiple records for the same name may have the same preference.

A host BAR.FOO.COM may want its mail to be delivered to the host PO.FOO.COM and would then use the MX record:

```
BAR.FOO.COM.     MX      10      PO.FOO.COM.
```

A host BAZ.FOO.COM may want its mail to be delivered to one of three different machines, in the following order:

```
BAZ.FOO.COM.     MX      10      PO1.FOO.COM.
                 MX      20      PO2.FOO.COM.
                 MX      30      PO3.FOO.COM.
```

An entire domain of hosts not connected to the Internet may want their mail to go through a mail gateway that knows how to deliver mail to them. If they would like mail addressed to any host in the domain FOO.COM to go through the mail gateway they might use:

```
FOO.COM.         MX       10     RELAY.CS.NET.
*.FOO.COM.       MX       20     RELAY.CS.NET.
```

Note that you can specify a wildcard in the MX record to match on anything in FOO.COM, but that it won't match a plain FOO.COM.

RFC-1033--Domain Operations Guide: Resource Records

IN-ADDR.ARPA

The structure of names in the domain system is set up in a hierarchical way such that the address of a name can be found by tracing down the domain tree contacting a server for each label of the name. Because of this 'indexing' based on name, there is no easy way to translate a host address back into its host name.

In order to do the reverse translation easily, a domain was created that uses hosts' addresses as part of a name that then points to the data for that host. In this way, there is now an 'index' to hosts' RRs based on their address. This address mapping domain is called `IN-ADDR.ARPA`. Within that domain are subdomains for each network, based on network number. Also, for consistency and natural groupings, the 4 octets of a host number are reversed.

For example, the ARPANET is net 10. That means there is a domain called `10.IN-ADDR.ARPA`. Within this domain there is a PTR RR at `51.0.0.10.IN-ADDR` that points to the RRs for the host `SRI-NIC.ARPA` (who's address is `10.0.0.51`). Since the NIC is also on the MILNET (Net 26, address `26.0.0.73`), there is also a PTR RR at `73.0.0.26.IN- ADDR.ARPA` that points to the same RR's for `SRI-NIC.ARPA`. The format of these special pointers is defined below along with the examples for the NIC.

RFC-1033--Domain Operations Guide: Resource Records

PTR

<special-name> [<ttl>] [<class>] PTR <name>

The PTR record is used to let special names point to some other location in the domain tree. They are mainly used in the IN-ADDR.ARPA records for translation of addresses to names. PTR's should use official names and not aliases.

For example, host SRI-NIC.ARPA with addresses 10.0.0.51 and 26.0.0.73 would have the following records in the respective zone files for net 10 and net 26:

```
51.0.0.10.IN-ADDR.ARPA.   PTR   SRI-NIC.ARPA.
73.0.0.26.IN-ADDR.ARPA.   PTR   SRI-NIC.ARPA.
```

Gateway PTR's

The IN-ADDR tree is also used to locate gateways on a particular network. Gateways have the same kind of PTR RRs as hosts (as above) but in addition they have other PTRs used to locate them by network number alone. These records have only 1, 2, or 3 octets as part of the name depending on whether they are class A, B, or C networks, respectively.

Lets take the SRI-CSL gateway for example. It connects 3 different networks, one class A, one class B and one class C. It will have the standard RR's for a host in the `CSL.SRI.COM` zone:

```
GW.CSL.SRI.COM.     A     10.2.0.2
                    A     128.18.1.1
                    A     192.12.33.2
```

Also, in 3 different zones (one for each network), it will have one of the following number to name translation pointers:

```
2.0.2.10.IN-ADDR.ARPA.       PTR   GW.CSL.SRI.COM.
1.1.18.128.IN-ADDR.ARPA.     PTR   GW.CSL.SRI.COM.
1.33.12.192.IN-ADDR.ARPA.    PTR   GW.CSL.SRI.COM.
```

In addition, in each of the same 3 zones will be one of the following gateway location pointers:

```
10.IN-ADDR.ARPA.             PTR   GW.CSL.SRI.COM.
18.128.IN-ADDR.ARPA.         PTR   GW.CSL.SRI.COM.
33.12.192.IN-ADDR.ARPA.      PTR   GW.CSL.SRI.COM.
```

RFC-1033--Domain Administrators Operations Guide

Instructions

Adding A Subdomain.

To add a new subdomain to your domain:

Setup the other domain server and/or the new zone file.

Add an NS record for each server of the new domain to the zone file of the parent domain.

Add any necessary glue RRs.

Adding A Host.

To add a new host to your zone files:

Edit the appropriate zone file for the domain the host is in.

Add an entry for each address of the host.

Optionally add CNAME, HINFO, WKS, and MX records.

Add the reverse IN-ADDR entry for each host address in the appropriate zone files for each network the host in on.

Deleting A Host.

To delete a host from the zone files:

Remove all the hosts' resource records from the zone file of the domain the host is in.

Remove all the hosts' PTR records from the IN-ADDR zone files for each network the host was on.

Also delete the gateway location PTR records for each network the gateway was on.

Complaints

These are the suggested steps you should take if you are having problems that you believe are caused by someone else's name server:

1. Complain privately to the responsible person for the domain. You can find their mailing address in the SOA record for the domain.

2. Complain publicly to the responsible person for the domain.

3. Ask the NIC for the administrative person responsible for the domain. Complain. You can also find domain contacts on the NIC in the file `NETINFO:DOMAIN-CONTACTS.TXT`

4. Complain to the parent domain authorities.

5. Ask the parent authorities to excommunicate the domain.

Example Domain Server Database Files

The following examples show how zone files are set up for a typical organization. SRI will be used as the example organization. SRI has decided to divided their domain `SRI.COM` into a few subdomains, one for each group that wants one. The subdomains are CSL and ISTC.

Note the following interesting items:

There are both hosts and domains under `SRI.COM`.

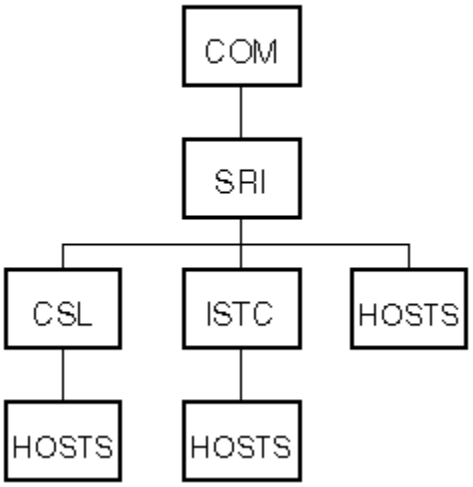
`CSL.SRI.COM` is both a domain name and a host name.

All the domains are serviced by the same pair of domain servers.

All hosts at SRI are on net `128.18` except hosts in the CSL domain which are on net `192.12.33`. Note that a domain does not have to correspond to a physical network.

The examples do not necessarily correspond to actual data in use by the SRI domain.

SRI Domain Organization



[File "`CONFIG.CMD`". Since bootstrap files are not standardized, this file is presented using a pseudo configuration file syntax.]

```
load root server list             from file ROOT.SERVERS
load zone SRI.COM.                from file SRI.ZONE
load zone CSL.SRI.COM.            from file CSL.ZONE
load zone ISTC.SRI.COM.           from file ISTC.ZONE
load zone 18.128.IN-ADDR.ARPA.    from file SRINET.ZONE
load zone 33.12.192.IN-ADDR.ARPA. from file SRI-CSL-NET.ZONE
```

[File "`ROOT.SERVERS`". Again, the format of this file is not standardized.]

```
;list of possible root servers
SRI-NIC.ARPA       10.0.0.51    26.0.0.73
C.ISI.EDU          10.0.0.52
```

```
BRL-AOS.ARPA        192.5.25.82  192.5.22.82   128.20.1.2
A.ISI.EDU           26.3.0.103
```

[File "SRI.ZONE"]

```
SRI.COM.        IN      SOA     KL.SRI.COM. DLE.STRIPE.SRI.COM. (
                                870407  ;serial
                                1800    ;refresh every 30 minutes
                                600     ;retry every 10 minutes
                                604800  ;expire after a week
                                86400   ;default of an hour
                                )

SRI.COM.        NS      KL.SRI.COM.
                NS      STRIPE.SRI.COM.
                MX      10      KL.SRI.COM.

;SRI.COM hosts

KL              A       10.1.0.2
                A       128.18.10.6
                MX      10      KL.SRI.COM.

STRIPE          A       10.4.0.2
STRIPE          A       128.18.10.4
                MX      10      STRIPE.SRI.COM.

NIC             CNAME   SRI-NIC.ARPA.

Blackjack       A       128.18.2.1
                HINFO   VAX-11/780      UNIX
                WKS     128.18.2.1      TCP TELNET FTP

CSL             A       192.12.33.2
                HINFO   FOONLY-F4       TOPS20
                WKS     192.12.33.2     TCP TELNET FTP SMTP FINGER
                MX      10      CSL.SRI.COM.
```

[File "CSL.ZONE"]

```
CSL.SRI.COM.    IN      SOA     KL.SRI.COM. DLE.STRIPE.SRI.COM. (
                                870330  ;serial
                                1800    ;refresh every 30 minutes
                                600     ;retry every 10 minutes
                                604800  ;expire after a week
                                86400   ;default of a day
                                )

CSL.SRI.COM.    NS              KL.SRI.COM.
                NS              STRIPE.SRI.COM.
                A               192.12.33.2

;CSL.SRI.COM hosts

A               CNAME   CSL.SRI.COM.
B               A       192.12.33.3
```

```
                HINFO   FOONLY-F4       TOPS20
                WKS     192.12.33.3     TCP TELNET FTP SMTP
GW              A       10.2.0.2
                A       192.12.33.1
                A       128.18.1.1
                HINFO   PDP-11/23       MOS
SMELLY          A       192.12.33.4
                HINFO   IMAGEN          IMAGEN
SQUIRREL        A       192.12.33.5
                HINFO   XEROX-1100      INTERLISP
VENUS           A       192.12.33.7
                HINFO   SYMBOLICS-3600  LISPM
HELIUM          A       192.12.33.30
                HINFO   SUN-3/160       UNIX
ARGON           A       192.12.33.31
                HINFO   SUN-3/75        UNIX
RADON           A       192.12.33.32
                HINFO   SUN-3/75        UNIX
```

[File "ISTC.ZONE"]

```
ISTC.SRI.COM.   IN  SOA     KL.SRI.COM. roemers.JOYCE.ISTC.SRI.COM. (
                            870406      ;serial
                            1800        ;refresh every 30 minutes
                            600         ;retry every 10 minutes
                            604800      ;expire after a week
                            86400       ;default of a day
                            )

ISTC.SRI.COM.   NS              KL.SRI.COM.
                NS              STRIPE.SRI.COM.
                MX              10      SPAM.ISTC.SRI.COM.

; ISTC hosts

joyce           A       128.18.4.2
                HINFO   VAX-11/750 UNIX
bozo            A       128.18.0.6
                HINFO   SUN UNIX
sundae          A       128.18.0.11
                HINFO   SUN UNIX
tsca            A       128.18.0.201
                A       10.3.0.2
                HINFO   VAX-11/750 UNIX
                MX      10  TSCA.ISTC.SRI.COM.
tsc             CNAME   tsca
prmh            A       128.18.0.203
                A       10.2.0.51
                HINFO   PDP-11/44 UNIX
spam            A       128.18.4.3
                A       10.2.0.107
                HINFO   VAX-11/780 UNIX
                MX      10  SPAM.ISTC.SRI.COM.
```

[File "SRINET.ZONE"]

```
18.128.IN-ADDR.ARPA.    IN  SOA  KL.SRI.COM  DLE.STRIPE.SRI.COM. (
                            870406  ;serial
                            1800    ;refresh every 30 minutes
                            600     ;retry every 10 minutes
                            604800  ;expire after a week
                            86400   ;default of a day
                            )

18.128.IN-ADDR.ARPA.    NS      KL.SRI.COM.
                        NS      STRIPE.SRI.COM.
                        PTR     GW.CSL.SRI.COM.

; SRINET [128.18.0.0] Address Translations

; SRI.COM Hosts
1.2.18.128.IN-ADDR.ARPA.        PTR     Blackjack.SRI.COM.

; ISTC.SRI.COM Hosts
2.4.18.128.IN-ADDR.ARPA.        PTR     joyce.ISTC.SRI.COM.
6.0.18.128.IN-ADDR.ARPA.        PTR     bozo.ISTC.SRI.COM.
11.0.18.128.IN-ADDR.ARPA.       PTR     sundae.ISTC.SRI.COM.
201.0.18.128.IN-ADDR.ARPA.      PTR     tsca.ISTC.SRI.COM.
203.0.18.128.IN-ADDR.ARPA.      PTR     prmh.ISTC.SRI.COM.
3.4.18.128.IN-ADDR.ARPA.        PTR     spam.ISTC.SRI.COM.

; CSL.SRI.COM Hosts
1.1.18.128.IN-ADDR.ARPA.        PTR     GW.CSL.SRI.COM.
```

[File "SRI-CSL-NET.ZONE"]

```
33.12.192.IN-ADDR.ARPA. IN  SOA KL.SRI.COM  DLE.STRIPE.SRI.COM. (
                            870404  ;serial
                            1800    ;refresh every 30 minutes
                            600     ;retry every 10 minutes
                            604800  ;expire after a week
                            86400   ;default of a day
                            )

33.12.192.IN-ADDR.ARPA. NS      KL.SRI.COM.
                        NS      STRIPE.SRI.COM.
                        PTR     GW.CSL.SRI.COM.

; SRI-CSL-NET [192.12.33.0] Address Translations

; SRI.COM Hosts
2.33.12.192.IN-ADDR.ARPA.       PTR     CSL.SRI.COM.

; CSL.SRI.COM Hosts
1.33.12.192.IN-ADDR.ARPA.       PTR     GW.CSL.SRI.COM.
3.33.12.192.IN-ADDR.ARPA.       PTR     B.CSL.SRI.COM.
4.33.12.192.IN-ADDR.ARPA.       PTR     SMELLY.CSL.SRI.COM.
5.33.12.192.IN-ADDR.ARPA.       PTR     SQUIRREL.CSL.SRI.COM.
7.33.12.192.IN-ADDR.ARPA.       PTR     VENUS.CSL.SRI.COM.
30.33.12.192.IN-ADDR.ARPA.      PTR     HELIUM.CSL.SRI.COM.
31.33.12.192.IN-ADDR.ARPA.      PTR     ARGON.CSL.SRI.COM.
32.33.12.192.IN-ADDR.ARPA.      PTR     RADON.CSL.SRI.COM.
```

RFC-1033--Domain Administrators Operations Guide: Appendix

BIND Files

BIND (Berkeley Internet Name Domain server) distributed with 4.3 BSD UNIX

This section describes two BIND implementation specific files; the boot file and the cache file. BIND has other options, files, and specifications that are not described here. See the Name Server Operations Guide for BIND for details.

The boot file for BIND is usually called "named.boot". This corresponds to file "CONFIG.CMD" in the example section.

```
-------------------------------------------------------
  cache         .                          named.ca
  primary       SRI.COM                    SRI.ZONE
  primary       CSL.SRI.COM                CSL.ZONE
  primary       ISTC.SRI.COM               ISTC.ZONE
  primary       18.128.IN-ADDR.ARPA        SRINET.ZONE
  primary       33.12.192.IN-ADDR.ARPA     SRI-CSL-NET.ZONE
  -------------------------------------------------------
```

The cache file for BIND is usually called "`named.ca`". This corresponds to file "`ROOT.SERVERS`" in the example section.

```
  ------------------------------------------------
  ;list of possible root servers
  .       1          IN   NS   SRI-NIC.ARPA.
                          NS   C.ISI.EDU.
                          NS   BRL-AOS.ARPA.
                          NS   C.ISI.EDU.
  ;and their addresses
  SRI-NIC.ARPA.           A    10.0.0.51
                          A    26.0.0.73
  C.ISI.EDU.              A    10.0.0.52
  BRL-AOS.ARPA.           A    192.5.25.82
                          A    192.5.22.82
                          A    128.20.1.2
  A.ISI.EDU.              A    26.3.0.103
  ------------------------------------------------
```

Dunlap, K., "Name Server Operations Guide for BIND", CSRG, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, California.

RFC-1034/RFC-1035: Domain Name System

P. Mockapetris
ISI
November 1987

Domain Name System -- RFC-1034 and RFC-1035

RFC-1034 describes the domain style names and their use for host address look up and electronic mail forwarding. It discusses the clients and servers in the domain name system and the protocol used between them. RFC-1035 documents the details of the domain name client - server communication.

These two RFCs have been merged in this document, under a single Table of Contents. RFC-1183, "New DNS RR Definitions" has also been integrated under the resource record section.

Other related RFCs are listed below:

- RFC-821 entitled "Simple Mail Transfer Protocol", and its companion, RFC-822 entitled "Standard for ARPA Internet Text Messages" specify a protocol and syntax for text messages within the framework of "electronic mail".

- RFC-974 entitled "Mail routing and the domain system" details the transition from HOSTS.TXT mail addressing to MX system.

- RFC-1031 entitled "MILNET Name Domain Transition" describes a plan for converting the MILNET to the DNS.

- RFC-1032 entitled "Establishing a Domain -- Guidelines for Administrators" discusses NIC registration policies.

- RFC-1033 entitled "Domain Administrators Operations Guide" provides guidelines for operating a domain server.

RFC-1034/5 Domain Name System

Introduction

This section introduces domain style names, their use for Internet mail and host address support, and the protocols and servers used to implement domain name facilities.

RFC-1034/5 Domain Name System: Introduction

Overview

The goal of domain names is to provide a mechanism for naming resources in such a way that the names are usable in different hosts, networks, protocol families, internets, and administrative organizations.

From the user's point of view, domain names are useful as arguments to a local agent, called a resolver, which retrieves information associated with the domain name. Thus a user might ask for the host address or mail information associated with a particular domain name. To enable the user to request a particular type of information, an appropriate query type is passed to the resolver with the domain name. To the user, the domain tree is a single information space; the resolver is responsible for hiding the distribution of data among name servers from the user.

From the resolver's point of view, the database that makes up the domain space is distributed among various name servers. Different parts of the domain space are stored in different name servers, although a particular data item will be stored redundantly in two or more name servers. The resolver starts with knowledge of at least one name server. When the resolver processes a user query it asks a known name server for the information; in return, the resolver either receives the desired information or a referral to another name server. Using these referrals, resolvers learn the identities and contents of other name servers. Resolvers are responsible for dealing with the distribution of the domain space and dealing with the effects of name server failure by consulting redundant databases in other servers.

Name servers manage two kinds of data. The first kind of data held in sets called zones; each zone is the complete database for a particular "pruned" subtree of the domain space. This data is called authoritative. A name server periodically checks to make sure that its zones are up to date, and if not, obtains a new copy of updated zones from master files stored locally or in another name server. The second kind of data is cached data which was acquired by a local resolver. This data may be incomplete, but improves the performance of the retrieval process when non-local data is repeatedly accessed. Cached data is eventually discarded by a timeout mechanism.

This functional structure isolates the problems of user interface, failure recovery, and distribution in the resolvers and isolates the database update and refresh problems in the name servers.

RFC-1034/5 Domain Name System: Introduction

The History of Domain Names

The impetus for the development of the domain system was growth in the Internet:

- Host name to address mappings were maintained by the Network Information Center (NIC) in a single file (HOSTS.TXT) which was FTPed by all hosts (RFC-952, RFC-953). The total network bandwidth consumed in distributing a new version by this scheme is proportional to the square of the number of hosts in the network, and even when multiple levels of FTP are used, the outgoing FTP load on the NIC host is considerable. Explosive growth in the number of hosts didn't bode well for the future.

- The network population was also changing in character. The timeshared hosts that made up the original ARPANET were being replaced with local networks of workstations. Local organizations were administering their own names and addresses, but had to wait for the NIC to change HOSTS.TXT to make changes visible to the Internet at large. Organizations also wanted some local structure on the name space.

- The applications on the Internet were getting more sophisticated and creating a need for general purpose name service.

The result was several ideas about name spaces and their management (IEN-116,,RFC-799, RFC-819, RFC-830). The proposals varied, but a common thread was the idea of a hierarchical name space, with the hierarchy roughly corresponding to organizational structure, and names using "." as the character to mark the boundary between hierarchy levels. A design using a distributed database and generalized resources was described in RFC-882, RFC-883. Based on experience with several implementations, the system evolved into the scheme described in this memo.

The terms "domain" or "domain name" are used in many contexts beyond the DNS described here. Very often, the term domain name is used to refer to a name with structure indicated by dots, but no relation to the DNS. This is particularly true in mail addressing (Quarterman 86).

RFC-1034/5 Domain Name System: Introduction

DNS Design Goals

The design goals of the DNS influence its structure. They are:

- The primary goal is a consistent name space which will be used for referring to resources. In order to avoid the problems caused by ad hoc encodings, names should not be required to contain network identifiers, addresses, routes, or similar information as part of the name.

- The sheer size of the database and frequency of updates suggest that it must be maintained in a distributed manner, with local caching to improve performance. Approaches that attempt to collect a consistent copy of the entire database will become more and more expensive and difficult, and hence should be avoided. The same principle holds for the structure of the name space, and in particular mechanisms for creating and deleting names; these should also be distributed.

- Where there tradeoffs between the cost of acquiring data, the speed of updates, and the accuracy of caches, the source of the data should control the tradeoff.

- The costs of implementing such a facility dictate that it be generally useful, and not restricted to a single application. We should be able to use names to retrieve host addresses, mailbox data, and other as yet undetermined information. All data associated with a name is tagged with a type, and queries can be limited to a single type.

- Because we want the name space to be useful in dissimilar networks and applications, we provide the ability to use the same name space with different protocol families or management. For example, host address formats differ between protocols, though all protocols have the notion of address. The DNS tags all data with a class as well as the type, so that we can allow parallel use of different formats for data of type address.

- We want name server transactions to be independent of the communications system that carries them. Some systems may wish to use datagrams for queries and responses, and only establish virtual circuits for transactions that need the reliability (e.g., database updates, long transactions); other systems will use virtual circuits exclusively.

- The system should be useful across a wide spectrum of host capabilities. Both personal computers and large timeshared hosts should be able to use the system, though perhaps in different ways.

Assumptions About Usage

The organization of the domain system derives from some assumptions about the needs and usage patterns of its user community and is designed to avoid many of the the complicated problems found in general purpose database systems.

The assumptions are:

- The size of the total database will initially be proportional to the number of hosts using the system, but will eventually grow to be proportional to the number of users on those hosts as mailboxes and other information are added to the domain system.

- Most of the data in the system will change very slowly (e.g., mailbox bindings, host addresses), but that the system should be able to deal with subsets that change more rapidly (on the order of seconds or minutes).

- The administrative boundaries used to distribute responsibility for the database will usually correspond to organizations that have one or more hosts. Each organization that has responsibility for a particular set of domains will provide redundant name servers, either on the organization's own hosts or other hosts that the organization arranges to use.

- Clients of the domain system should be able to identify trusted name servers they prefer to use before accepting referrals to name servers outside of this "trusted" set.

- Access to information is more critical than instantaneous updates or guarantees of consistency. Hence the update process allows updates to percolate out through the users of the domain system rather than guaranteeing that all copies are simultaneously updated. When updates are unavailable due to network or host failure, the usual course is to believe old information while continuing efforts to update it. The general model is that copies are distributed with timeouts for refreshing. The distributor sets the timeout value and the recipient of the distribution is responsible for performing the refresh. In special situations, very short intervals can be specified, or the owner can prohibit copies.

- In any system that has a distributed database, a particular name server may be presented with a query that can only be answered by some other server. The two general approaches to dealing with this problem are "recursive", in which the first server pursues the query for the client at another server, and "iterative", in which the server refers the client to another server and lets the client pursue the query. Both approaches have advantages and disadvantages, but the iterative approach is preferred for the datagram style of access. The domain system requires implementation of the iterative approach, but allows the recursive approach as an option.

The domain system assumes that all data originates in master files scattered through the

hosts that use the domain system. These master files are updated by local system administrators. Master files are text files that are read by a local name server, and hence become available through the name servers to users of the domain system. The user programs access name servers through standard programs called resolvers.

The standard format of master files allows them to be exchanged between hosts (via FTP, mail, or some other mechanism); this facility is useful when an organization wants a domain, but doesn't want to support a name server. The organization can maintain the master files locally using a text editor, transfer them to a foreign host which runs a name server, and then arrange with the system administrator of the name server to get the files loaded.

Each host's name servers and resolvers are configured by a local system administrator (see RFC-1033 Domain Administrators Guide). For a name server, this configuration data includes the identity of local master files and instructions on which non-local master files are to be loaded from foreign servers. The name server uses the master files or copies to load its zones. For resolvers, the configuration data identifies the name servers which should be the primary sources of information.

The domain system defines procedures for accessing the data and for referrals to other name servers. The domain system also defines procedures for caching retrieved data and for periodic refreshing of data defined by the system administrator.

The system administrators provide:

- The definition of zone boundaries.
- Master files of data.
- Updates to master files.
- Statements of the refresh policies desired.

The domain system provides:

- Standard formats for resource data.
- Standard methods for querying the database.
- Standard methods for name servers to refresh local data from foreign name servers.

RFC-1034/5 Domain Name System: Introduction

Elements of the DNS

The DNS has three major components:

- The Domain Name Space and Resource Records, which are specifications for a tree structured name space and data associated with the names. Conceptually, each node and leaf of the domain name space tree names a set of information, and query operations are attempts to extract specific types of information from a particular set. A query names the domain name of interest and describes the type of resource information that is desired. For example, the Internet uses some of its domain names to identify hosts; queries for address resources return Internet host addresses.

- Name Servers are server programs which hold information about the domain tree's structure and set information. A name server may cache structure or set information about any part of the domain tree, but in general a particular name server has complete information about a subset of the domain space, and pointers to other name servers that can be used to lead to information from any part of the domain tree. Name servers know the parts of the domain tree for which they have complete information; a name server is said to be an AUTHORITY for these parts of the name space. Authoritative information is organized into units called ZONEs, and these zones can be automatically distributed to the name servers which provide redundant service for the data in a zone.

- RESOLVERS are programs that extract information from name servers in response to client requests. Resolvers must be able to access at least one name server and use that name server's information to answer a query directly, or pursue the query using referrals to other name servers. A resolver will typically be a system routine that is directly accessible to user programs; hence no protocol is necessary between the resolver and the user program.

These three components roughly correspond to the three layers or views of the domain system:

- From the user's point of view, the domain system is accessed through a simple procedure or OS call to a local resolver. The domain space consists of a single tree and the user can request information from any section of the tree.

- From the resolver's point of view, the domain system is composed of an unknown number of name servers. Each name server has one or more pieces of the whole domain tree's data, but the resolver views each o1f these databases as essentially static.

- From a name server's point of view, the domain system consists of separate sets of local information called zones. The name server has local copies of some of the zones. The name server must periodically refresh its zones from master copies in local files or foreign name servers. The name server must concurrently process queries that

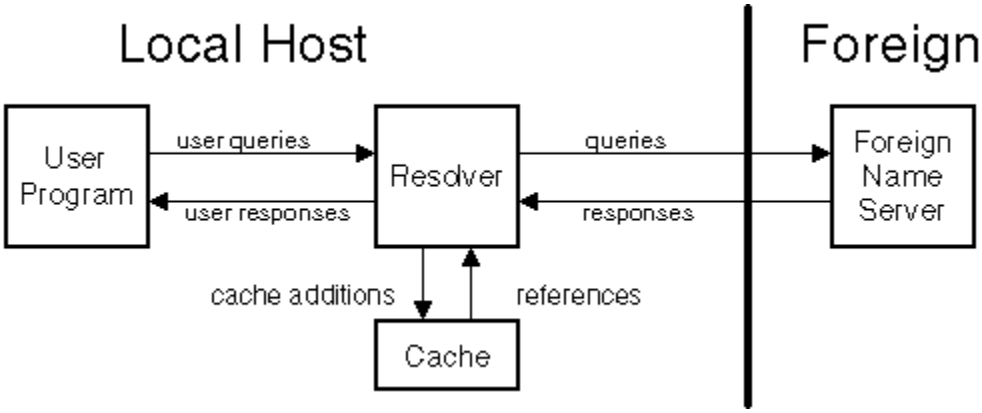
arrive from resolvers.

In the interests of performance, implementations may couple these functions. For example, a resolver on the same machine as a name server might share a database consisting of the the zones managed by the name server and the cache managed by the resolver.

RFC-1034/5 Domain Name System: Introduction

Common Configurations

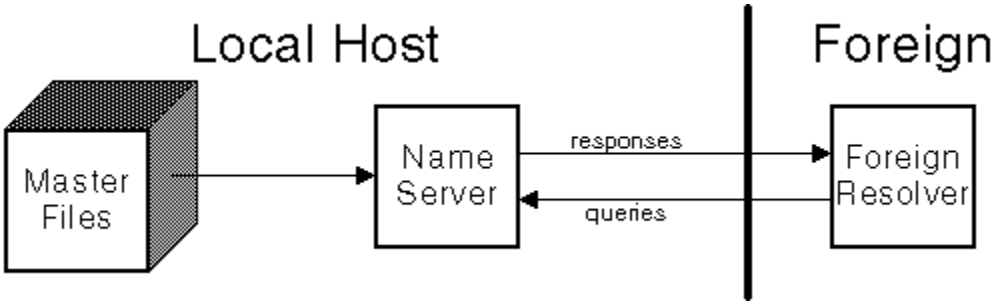
A host can participate in the domain name system in a number of ways, depending on whether the host runs programs that retrieve information from the domain system, name servers that answer queries from other hosts, or various combinations of both functions. The simplest, and perhaps most typical, configuration is shown below:



User programs interact with the domain name space through resolvers; the format of user queries and user responses is specific to the host and its operating system. User queries will typically be operating system calls, and the resolver and its cache will be part of the host operating system. Less capable hosts may choose to implement the resolver as a subroutine to be linked in with every program that needs its services. Resolvers answer user queries with information they acquire via queries to foreign name servers and the local cache.

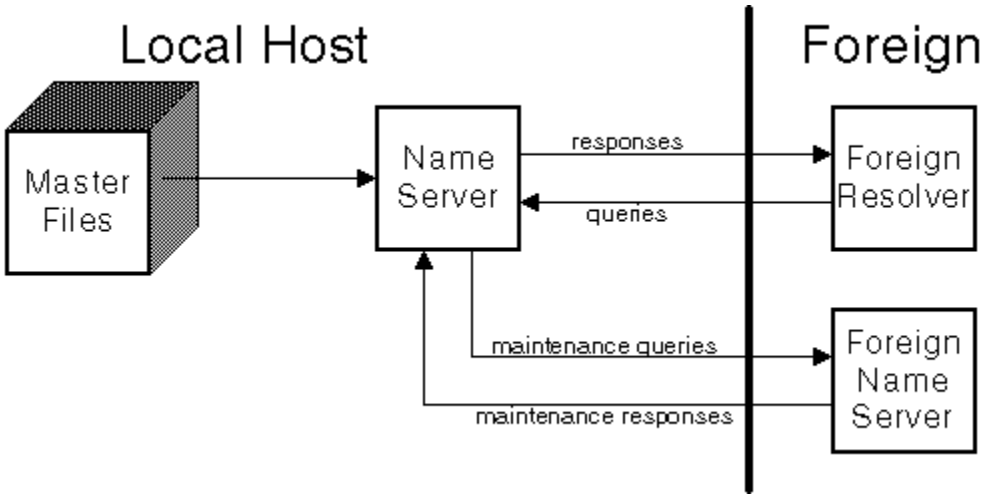
Note that the resolver may have to make several queries to several different foreign name servers to answer a particular user query, and hence the resolution of a user query may involve several network accesses and an arbitrary amount of time. The queries to foreign name servers and the corresponding responses have a standard format described in this memo, and may be datagrams.

Depending on its capabilities, a name server could be a stand alone program on a dedicated machine or a process or processes on a large timeshared host. A simple configuration might be:



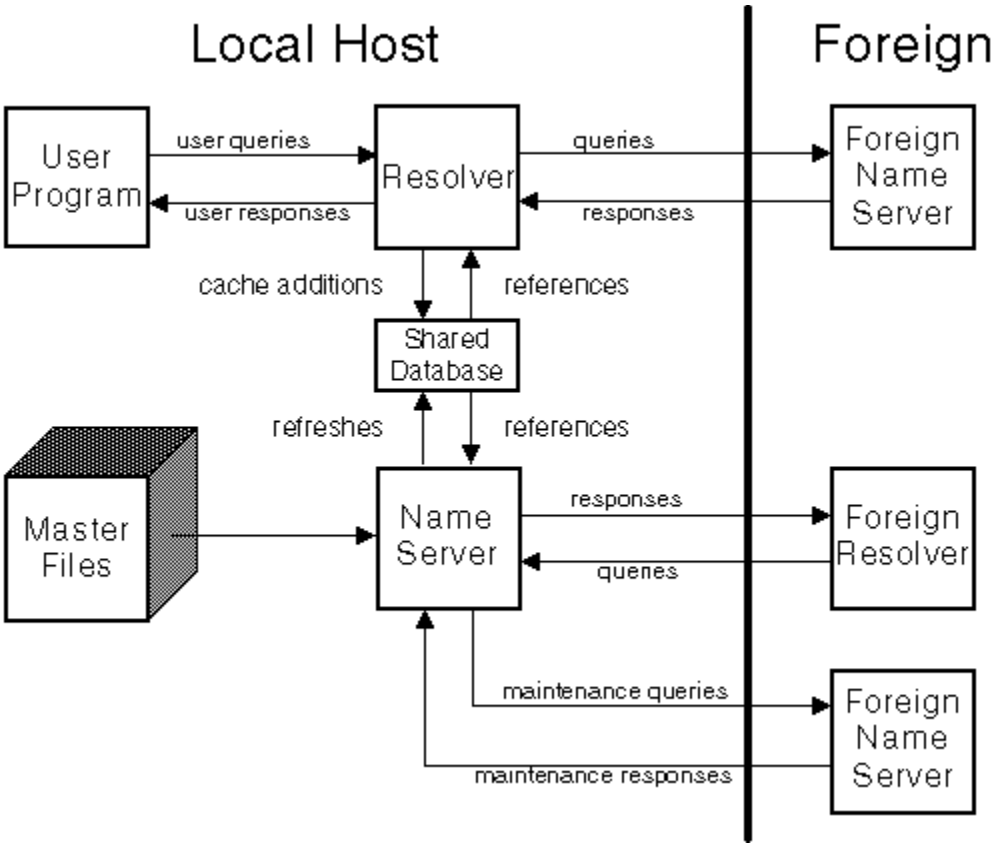
Here a primary name server acquires information about one or more zones by reading master files from its local file system, and answers queries about those zones that arrive from foreign resolvers.

The DNS requires that all zones be redundantly supported by more than one name server. Designated secondary servers can acquire zones and check for updates from the primary server using the zone transfer protocol of the DNS. This configuration is shown below:

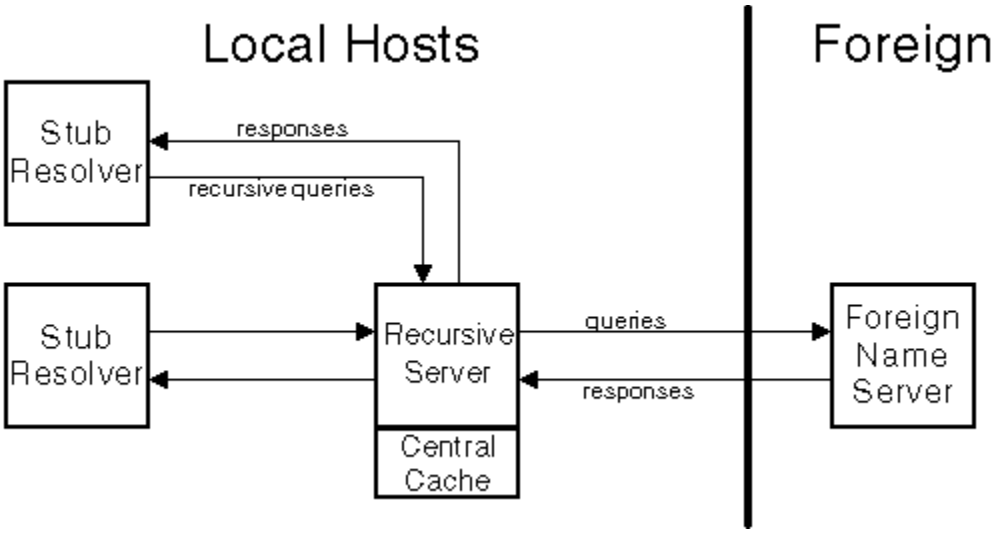


In this configuration, the name server periodically establishes a virtual circuit to a foreign name server to acquire a copy of a zone or to check that an existing copy has not changed. The messages sent for these maintenance activities follow the same form as queries and responses, but the message sequences are somewhat different.

The information flow in a host that supports all aspects of the domain name system is shown below:



The shared database holds domain space data for the local name server and resolver. The contents of the shared database will typically be a mixture of authoritative data maintained by the periodic refresh operations of the name server and cached data from previous resolver requests. The structure of the domain data and the necessity for synchronization between name servers and resolvers imply the general characteristics of this database, but the actual format is up to the local implementor. Information flow can also be tailored so that a group of hosts act together to optimize activities. Sometimes this is done to offload less capable hosts so that they do not have to implement a full resolver. This can be appropriate for PCs or hosts which want to minimize the amount of new network code which is required. This scheme can also allow a group of hosts can share a small number of caches rather than maintaining a large number of separate caches, on the premise that the centralized caches will have a higher hit ratio. In either case, resolvers are replaced with stub resolvers which act as front ends to resolvers located in a recursive server in one or more name servers known to perform that service:



In any case, note that domain components are always replicated for reliability whenever possible.

Conventions

The domain system has several conventions dealing with low-level, but fundamental, issues. While the implementor is free to violate these conventions WITHIN HIS OWN SYSTEM, he must observe these conventions in ALL behavior observed from other hosts.

RFC-1034/5 Domain Name System: Conventions

Preferred Name Syntax

The DNS specifications attempt to be as general as possible in the rules for constructing domain names. The idea is that the name of any existing object can be expressed as a domain name with minimal changes. However, when assigning a domain name for an object, the prudent user will select a name which satisfies both the rules of the domain system and any existing rules for the object, whether these rules are published or implied by existing programs.

For example, when naming a mail domain, the user should satisfy both the rules of this memo and those in RFC-822 describing mail format. When creating a new host name, the old rules for HOSTS.TXT should be followed. This avoids problems when old software is converted to use domain names.

The following syntax will result in fewer problems with many applications that use domain names (e.g., mail, TELNET).

```
<domain> ::= <subdomain> | " "

<subdomain> ::= <label> | <subdomain> "." <label>

<label> ::= <letter> [ [ <ldh-str> ] <let-dig> ]

<ldh-str> ::= <let-dig-hyp> | <let-dig-hyp> <ldh-str>

<let-dig-hyp> ::= <let-dig> | "-"

<let-dig> ::= <letter> | <digit>

<letter> ::= any one of the 52 alphabetic characters A through Z in upper case
and a through z in lower case

<digit> ::= any one of the ten digits 0 through 9
```

Note that while upper and lower case letters are allowed in domain names, no significance is attached to the case. That is, two names with the same spelling but different case are to be treated as if identical.

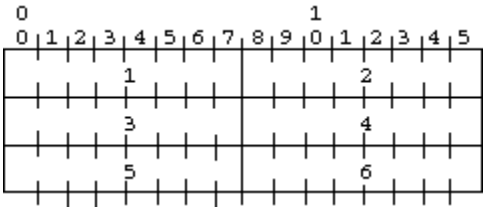
The labels must follow the rules for ARPANET host names. They must start with a letter, end with a letter or digit, and have as interior characters only letters, digits, and hyphen. There are also some restrictions on the length. Labels must be 63 characters or less.

For example, the following strings identify hosts in the Internet:

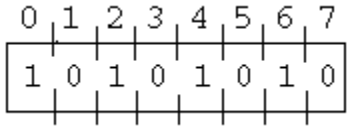
A.ISI.EDU XX.LCS.MIT.EDU SRI-NIC.ARPA

Data Transmission Order

The order of transmission of the header and data described in this document is resolved to the octet level. Whenever a diagram shows a group of octets, the order of transmission of those octets is the normal order in which they are read in English. For example, in the following diagram, the octets are transmitted in the order they are numbered.



Whenever an octet represents a numeric quantity, the left most bit in the diagram is the high order or most significant bit. That is, the bit labeled 0 is the most significant bit. For example, the following diagram represents the value 170 (decimal).



Similarly, whenever a multi-octet field represents a numeric quantity the left most bit of the whole field is the most significant bit. When a multi-octet quantity is transmitted the most significant octet is transmitted first.

RFC-1034/5 Domain Name System: Conventions

Character Case

For all parts of the DNS that are part of the official protocol, all comparisons between character strings (e.g., labels, domain names, etc.) are done in a case-insensitive manner. At present, this rule is in force throughout the domain system without exception. However, future additions beyond current usage may need to use the full binary octet capabilities in names, so attempts to store domain names in 7-bit ASCII or use of special bytes to terminate labels, etc., should be avoided.

When data enters the domain system, its original case should be preserved whenever possible. In certain circumstances this cannot be done. For example, if two RRs are stored in a database, one at `x.y` and one at `X.Y`, they are actually stored at the same place in the database, and hence only one casing would be preserved. The basic rule is that case can be discarded only when data is used to define structure in a database, and two names are identical when compared in a case insensitive manner.

Loss of case sensitive data must be minimized. Thus while data for `x.y` and `X.Y` may both be stored under a single location x.y or `X.Y`, data for `a.x` and `B.X` would never be stored under `A.x`, `A.X`, `b.x`, or `b.X`. In general, this preserves the case of the first label of a domain name, but forces standardization of interior node labels.

Systems administrators who enter data into the domain database should take care to represent the data they supply to the domain system in a case-consistent manner if their system is case-sensitive. The data distribution system in the domain system will ensure that consistent representations are preserved.

Size limits

Various objects and parameters in the DNS have size limits. They are listed below. Some could be easily changed, others are more fundamental.

labels	63 octets or less
names	255 octets or less
TTL	positive values of a signed 32 bit number.
UDP messages	512 octets or less

RFC-1034/5 Domain Name System

Domain Name Space and Resource Records

RFC-1034/5 Domains: Name Space and Resource Records

Name Space Specifications and Terminology

The domain name space is a tree structure. Each node and leaf on the tree corresponds to a resource set (which may be empty). The domain system makes no distinctions between the uses of the interior nodes and leaves, and this memo uses the term "node" to refer to both.

Each node has a label, which is zero to 63 octets in length. Brother nodes may not have the same label, although the same label can be used for nodes which are not brothers. One label is reserved, and that is the null (i.e., zero length) label used for the root.

The domain name of a node is the list of the labels on the path from the node to the root of the tree. By convention, the labels that compose a domain name are printed or read left to right, from the most specific (lowest, farthest from the root) to the least specific (highest, closest to the root).

Internally, programs that manipulate domain names should represent them as sequences of labels, where each label is a length octet followed by an octet string.Because all domain names end at the root, which has a null string for a label, these internal representations can use a length byte of zero to terminate a domain name.

By convention, domain names can be stored with arbitrary case, but domain name comparisons for all present domain functions are done in a case-insensitive manner, assuming an ASCII character set, and a high order zero bit. This means that you are free to create a node with label "A" or a node with label "a", but notboth as brothers; you could refer to either using "a" or "A". When you receive a domain name or label, you should preserve its case. The rationale for this choice is that we may someday need to add full binary domain names for new services; existing services would not be changed.

When a user needs to type a domain name, the length of each label is omitted and the labels are separated by dots ("."). Since a complete domain name ends with the root label, this leads to a printed form which ends in a dot. We use this property to distinguish between:

- a character string which represents a complete domain name (often called "absolute"). For example, "`poneria.ISI.EDU.`"

- a character string that represents the starting labels of a domain name which is incomplete, and should be completed by local software using knowledge of the local domain (often called "relative"). For example, "`poneria`" used in the `ISI.EDU` domain.

Relative names are either taken relative to a well known origin, or to a list of domains used as a search list. Relative names appear mostly at the user interface, where their interpretation varies from implementation to implementation, and in master files, where they are relative to a single origin domain name. The most common interpretation uses the root "." as either the single origin or as one of the members of the search list, so a multi-label relative name is oftenone where the trailing dot has been omitted to save typing.

To simplify implementations, the total number of octets that represent a domain name (i.e., the sum of all label octets and label lengths) is limited to 255.

A domain is identified by a domain name, and consists of that part of the domain name

space that is at or below the domain name which specifies the domain. A domain is a subdomain of another domain if it is contained within that domain. This relationship can be tested by seeing if the subdomain's name ends with the containing domain's name. For example, `A.B.C.D` is a subdomain of `B.C.D`, `C.D`, `D`, and " ".

RFC-1034/5 Domains: Name Space and Resource Records

Name Space Definitions

Domain names in messages are expressed in terms of a sequence of labels. Each label is represented as a one octet length field followed by that number of octets. Since every domain name ends with the null label of the root, a domain name is terminated by a length byte of zero. The high order two bits of every length octet must be zero, and the remaining six bits of the length field limit the label to 63 octets or less.

To simplify implementations, the total length of a domain name (i.e., label octets and label length octets) is restricted to 255 octets or less.

Although labels can contain any 8 bit values in octets that make up a label, it is strongly recommended that labels follow the preferred syntax described elsewhere in this memo, which is compatible with existing host naming conventions. Name servers and resolvers must compare labels in a case-insensitive manner (i.e., A=a), assuming ASCII with zero parity. Non-alphabetic codes must match exactly.

Administrative Guidelines on Use

As a matter of policy, the DNS technical specifications do not mandate a particular tree structure or rules for selecting labels; its goal is to be as general as possible, so that it can be used to build arbitrary applications. In particular, the system was designed so that the name space did not have to be organized along the lines of network boundaries, name servers, etc. The rationale for this is not that the name space should have no implied semantics, but rather that the choice of implied semantics should be left open to be used for the problem at hand, and that different parts of the tree can have different implied semantics. For example, the `IN-ADDR.ARPA` domain is organized and distributed by network and host address because its role is to translate from network or host numbers to names; NetBIOS domains (RFC-1001, RFC- 1002) are flat because that is appropriate for that application.

However, there are some guidelines that apply to the "normal" parts of the name space used for hosts, mailboxes, etc., that will make the name space more uniform, provide for growth, and minimize problems as software is converted from the older host table. The political decisions about the top levels of the tree originated in RFC-920. Current policy for the top levels is discussed in RFC-1032, the Administrators Guide. MILNET conversion issues are covered in RFC-1031.

Lower domains which will eventually be broken into multiple zones should provide branching at the top of the domain so that the eventual decomposition can be done without renaming. Node labels which use special characters, leading digits, etc., are likely to break older software which depends on more restrictive choices.

RFC-1034/5 Domains: Name Space and Resource Records

Technical Guidelines on Use

Before the DNS can be used to hold naming information for some kind of object, two needs must be met:

- A convention for mapping between object names and domain names. This describes how information about an object is accessed.
- RR types and data formats for describing the object.

These rules can be quite simple or fairly complex. Very often, the designer must take into account existing formats and plan for upward compatibility for existing usage. Multiple mappings or levels of mapping may be required.

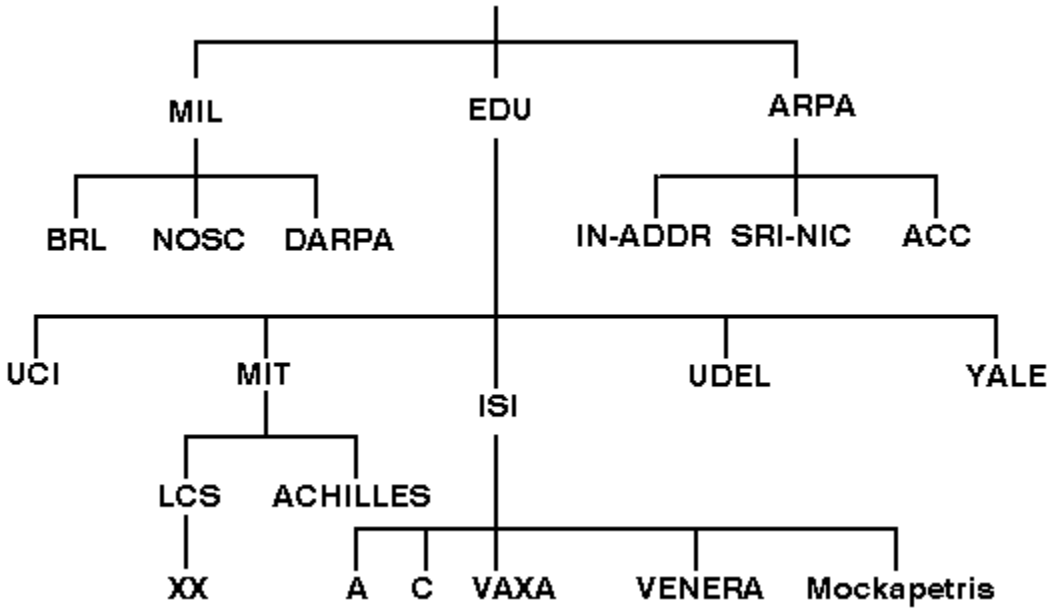
For hosts, the mapping depends on the existing syntax for host names which is a subset of the usual text representation for domain names, together with RR formats for describing host addresses, etc. Because we need a reliable inverse mapping from address to host name, a special mapping for addresses into the IN-ADDR.ARPA domain is also defined.

For mailboxes, the mapping is slightly more complex. The usual mail address <`local-part`>@<`mail-domain`> is mapped into a domain name by converting <`local-part`>into a single label (regardles of dots it contains), converting <`mail-domain`> into a domain name using the usual text format for domain names (dots denote label breaks), and concatenating the two to form a single domain name. Thus the mailbox `HOSTMASTER@SRI-NIC.ARP`A is represented as a domain name by `HOSTMASTER.SRI-NIC.ARPA`. An appreciation for the reasons behind this design also must take into account the scheme for **mail exchanges** (RFC- 974).

The typical user is not concerned with defining these rules, but should understand that they usually are the result of numerous compromises between desires forupward compatibility with old usage, interactions between different object definitions, and the inevitable urge to add new features when defining the rules. The way the DNS is used to support some object is often more crucial than the restrictions inherent in the DNS.

Example Name Space

The following figure shows a part of the current domain name space, and is used in many examples in this RFC. Note that the tree is a very small subset of the actual name space.



In this example, the root domain has three immediate subdomains: `MIL`, `EDU`, and `ARPA`. The `LCS.MIT.EDU` domain has one immediate subdomain named `XX.LCS.MIT.EDU`. All of the leaves are also domains.

Resource Records

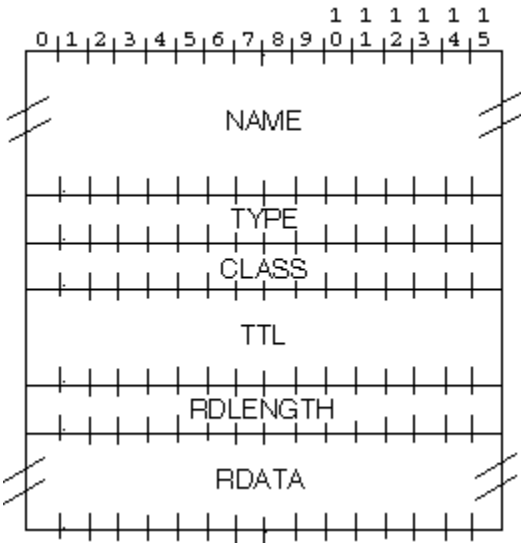
A domain name identifies a node. Each node has a set of resource information, which may be empty. The set of resource information associated with a particular name is composed of separate resource records (RRs). The order of RRs in a set is not significant, and need not be preserved by name servers, resolvers, or other parts of the DNS.

RR Definitions

RFC-1034/5 Domains: Name Space and Resource Records

Format

All RRs have the same top level format shown below:



where:

NAME an owner name, i.e., the name of the node to which this resource record pertains.

TYPE two octets containing one of the RR TYPE codes.

CLASS two octets containing one of the RR CLASS codes.

TTL a 32 bit signed integer that specifies the time interval that the resource record may be cached before the source of the information should again be consulted. Zero values are interpreted to mean that the RR can only be used for the transaction in progress, and should not be cached. For example, SOA records are always distributed with a zero TTL to prohibit caching. Zero values can also be used for extremely volatile data.

RDLENGTH an unsigned 16 bit integer that specifies the length in octets of the RDATA field.

RDATA a variable length string of octets that describes the resource. The format of this information varies according to the TYPE and CLASS of the resource record.

RFC-1034/5 Domains: Name Space and Resource Records

TYPE Values

TYPE fields are used in resource records. Note that these types are a subset of QTYPEs.

TYPE	value and meaning
A	1 a host address
NS	2 an authoritative name server
MD	3 a mail destination (Obsolete - use MX)
MF	4 a mail forwarder (Obsolete - use MX)
CNAME	5 the canonical name for an alias
SOA	6 marks the start of a zone of authority
MB	7 a mailbox domain name (EXPERIMENTAL)
MG	8 a mail group member (EXPERIMENTAL)
MR	9 a mail rename domain name (EXPERIMENTAL)
NULL	10 a null RR (EXPERIMENTAL)
WKS	11 a well known service description
PTR	12 a domain name pointer
HINFO	13 host information
MINFO	14 mailbox or mail list information
MX	15 mail exchange
TXT	16 text strings

QTYPE Values

QTYPE fields appear in the question part of a query. QTYPES are a superset of TYPEs, hence all TYPEs are valid QTYPEs. In addition, the following QTYPEs are defined:

AXFR	252 A request for a transfer of an entire zone
MAILB	253 A request for mailbox-related records (MB, MG or MR)
MAILA	254 A request for mail agent RRs (Obsolete - see MX)
*	255 A request for all records

CLASS Values

CLASS fields appear in resource records. The following CLASS mnemonics and values are defined:

IN	1	the Internet
CS	2	the CSNET class (Obsolete - used only for examples in some obsolete RFCs)
CH	3	the CHAOS class
HS	4	Hesiod {Dyer 87)

QCLASS Values

QCLASS fields appear in the question section of a query. QCLASS values are a superset of CLASS values; every CLASS is a valid QCLASS. In addition to CLASS values, the following QCLASSes are defined:

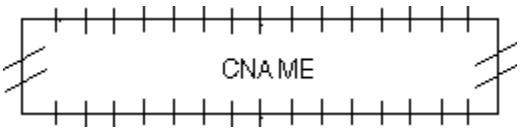
*	255 any class

Standard RRs

The following RR definitions are expected to occur, at least potentially, in all classes. In particular, NS, SOA, CNAME, and PTR will be used in all classes, and have the same format in all classes. Because their RDATA format is known, all domain names in the RDATA section of these RRs may be compressed.

<`domain-name`> is a domain name represented as a series of labels, and terminated by a label with zero length. <`character-string`> is a single length octet followed by that number of characters. <`character-string`> is treated as binary information, and can be up to 256 characters in length (including the length octet).

CNAME RDATA Format

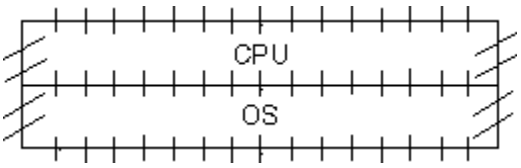


where:

CNAME A <domain-name> which specifies the canonical or primary name for the owner. The owner name is an alias.

CNAME RRs cause no additional section processing, but name servers may choose to restart the query at the canonical name in certain cases. See the description of name server logic for details.

HINFO RDATA Format



where:

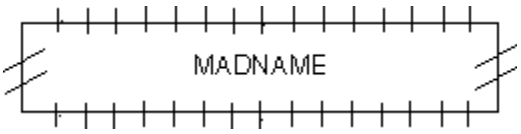
CPU A <character-string> which specifies the CPU type.

OS A <character-string> which specifies the operating system type.

Standard values for CPU and OS can be found in RFC-1010.

HINFO records are used to acquire general information about a host. The main use is for protocols such as FTP that can use special procedures when talking between machines or operating systems of the same type.

MB RDATA Format (EXPERIMENTAL)

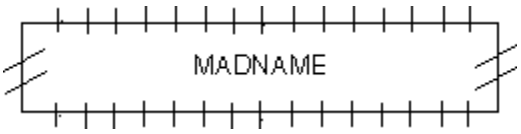


where:

MADNAME A <domain-name> which specifies a host which has the specified mailbox.

MB records cause additional section processing which looks up an A type RRs corresponding to MADNAME.

MD RDATA Format (Obsolete)



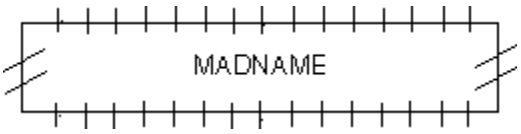
where:

MADNAME A <domain-name> which specifies a host which has a mail agent for the domain which should be able to deliver mail for the domain.

MD records cause additional section processing which looks up an A type record corresponding to MADNAME.

MD is obsolete. See the definition of MX and RFC-974 on mail routing for details of the new scheme. The recommended policy for dealing with MD RRs found in a master file is to reject them, or to convert them to MX RRs with a preference of 0.

MF RDATA Format (Obsolete)



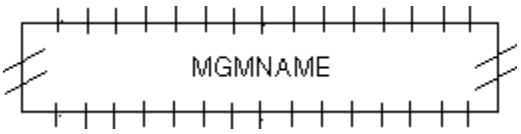
where:

MADNAME A <domain-name> which specifies a host which has a mail agent for the domain which will accept mail for forwarding to the domain.

MF records cause additional section processing which looks up an A type record corresponding to MADNAME.

MF is obsolete. See the definition of MX and RFC-974 on mail routingfor details of the new scheme. The recommended policy for dealing with MD RRs found in a master file is to reject them, or to convert them to MX RRs with a preference of 10.

MG RDATA Format (EXPERIMENTAL)

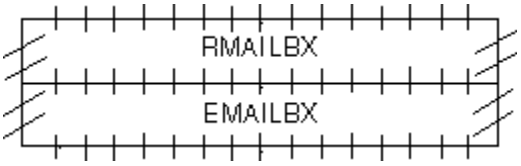


where:

MGMNAME A <`domain-name`> which specifies a mailbox which is a member of the mail group specified by the domain name.

MG records cause no additional section processing.

MINFO RDATA Format (EXPERIMENTAL)



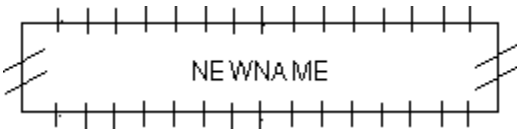
where:

RMAILBX A <domain-name> which specifies a mailbox which is responsible for the mailing list or mailbox. If this domain name names the root, the owner of the MINFO RR is responsible for itself. Note that many existing mailing lists use a mailbox X-request for the RMAILBX field of mailing list X, e.g., Msgroup-request for Msgroup. This field provides a more general mechanism.

EMAILBX A <domain-name> which specifies a mailbox which is to receive error messages related to the mailing list or mailbox specified by the owner of the MINFO RR (similar to the ERRORS-TO: field which has been proposed). If this domain name names the root, errors should be returned to the sender of the message.

MINFO records cause no additional section processing. Although these records can be associated with a simple mailbox, they are usually used with a mailing list.

MR RDATA Format (EXPERIMENTAL)



where:

NEWNAME A <domain-name> which specifies a mailbox which is the proper rename of the specified mailbox.

MR records cause no additional section processing. The main use for MR is as a forwarding entry for a user who has moved to a different mailbox.

MX RDATA Format

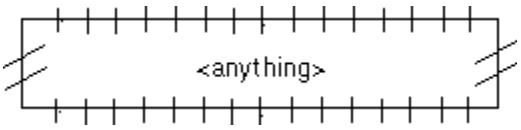
where:

PREFERENCE A 16 bit integer which specifies the preference given to this RR among others at the same owner. Lower values are preferred.

EXCHANGE A <`domain-name`> which specifies a host willing to act as a mail exchange for the owner name.

MX records cause type A additional section processing for the host specified by EXCHANGE. The use of MX RRs is explained in detail in RFC-974 on mail routing.

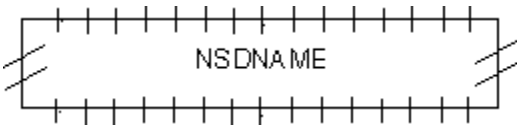
NULL RDATA Format (EXPERIMENTAL)



Anything at all may be in the RDATA field so long as it is 65535 octets or less.

NULL records cause no additional section processing. NULL RRs are not allowed in master files. NULLs are used as placeholders in some experimental extensions of the DNS.

NS RDATA Format



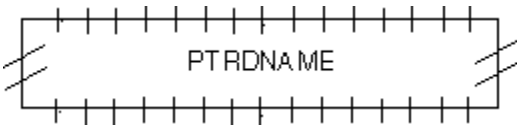
where:

NSDNAME A <domain-name> which specifies a host which should be authoritative for the specified class and domain.

NS records cause both the usual additional section processing to locate a type A record, and, when used in a referral, a special search of the zone in which they reside for glue information.

The NS RR states that the named host should be expected to have a zone starting at owner name of the specified class. Note that the class may not indicate the protocol family which should be used to communicate with the host, although it is typically a strong hint. For example, hosts which are name servers for either Internet (IN) or Hesiod (HS) class information are normally queried using IN class protocols.

PTR RDATA Format

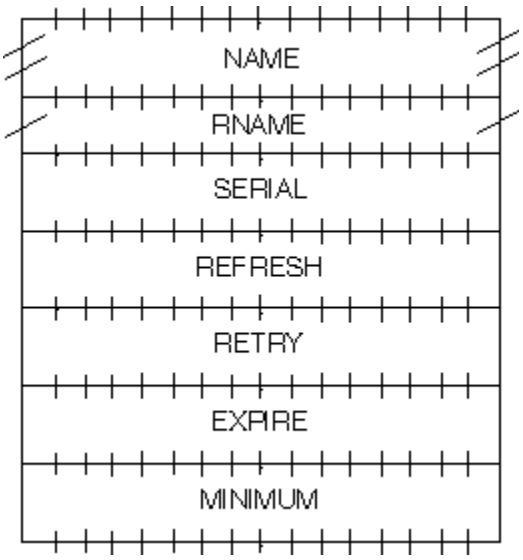


where:

PTRDNAME A <domain-name> which points to some location in the domain name space.

PTR records cause no additional section processing. These RRs are used in special domains to point to some other location in the domain space. These records are simple data, and don't imply any special processing similar to that performed by CNAME, which identifies aliases. See the description of the IN-ADDR.ARPA domain for an example.

SOA RDATA Format



where:

MNAME The <domain-name> of the name server that was the original or primary source of data for this zone.

RNAME A <domain-name> which specifies the mailbox of the person responsible for this zone.

SERIAL The unsigned 32 bit version number of the original copy of the zone. Zone transfers preserve this value. This value wraps and should be compared using sequence space arithmetic.

REFRESH A 32 bit time interval before the zone should be refreshed.

RETRY A 32 bit time interval that should elapse before a failed refresh should be retried.

EXPIRE A 32 bit time value that specifies the upper limit on the time interval that can elapse before the zone is no longer authoritative.

MINIMUM The unsigned 32 bit minimum TTL field that should be exported with any RR from this zone.

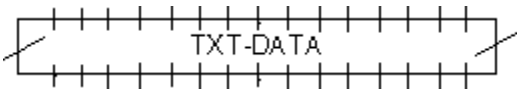
SOA records cause no additional section processing.

All times are in units of seconds.

Most of these fields are pertinent only for name server maintenance operations. However, MINIMUM is used in all query operations that retrieve RRs from a zone. Whenever a RR is sent in a response to a query, the TTL field is set to the maximum of the TTL field from the RR and the MINIMUM field in the appropriate SOA. Thus MINIMUM is a lower bound on the TTL field for all RRs in a zone. Note that this use of MINIMUM should occur when the RRs

are copied into the response and not when the zone is loaded from a master file or via a zone transfer. The reason for this provison is to allow future dynamic update facilities to change the SOA RR with known semantics.

TXT RDATA format



where:

TXT-DATA One or more <character-string>s.

TXT RRs are used to hold descriptive text. The semantics of the text depends on the domain where it is found.

Textual Expression of RRs

RRs are represented in binary form in the packets of the DNS protocol, and are usually represented in highly encoded form when stored in a name server or resolver. In this memo, we adopt a style similar to that used in master files in order to show the contents of RRs. In this format, most RRs are shown on a singleline, although continuation lines are possible using parentheses.

The start of the line gives the owner of the RR. If a line begins with a blank, then the owner is assumed to be the same as that of the previous RR. Blank lines are often included for readability.

In order to avoid ambiguity in parsing, type and class mnemonics are disjoint, TTLs are integers, and the type mnemonic is always last. The IN class and TTL values are often omitted from examples in the interests of clarity.

The resource data or RDATA section of the RR are given using knowledge of the typical representation for the data.

For example, we might show the RRs carried in a message as:

```
ISI.EDU.          MX      10 VENERA.ISI.EDU.
                  MX      10 VAXA.ISI.EDU.
VENERA.ISI.EDU.   A       128.9.0.32
                  A       10.1.0.52
VAXA.ISI.EDU.     A       10.2.0.27
                  A       128.9.0.33
```

The MX RRs have an RDATA section which consists of a 16 bit number followed by a domain name. The address RRs use a standard IP address format to contain a 32bit internet address.

This example shows six RRs, with two RRs at each of three domain names.

Similarly we might see:

```
 XX.LCS.MIT.EDU. IN      A       10.0.0.44
                 CH      A       MIT.EDU. 2420
```

This example shows two addresses for `XX.LCS.MIT.EDU`, each of a different class.

Master Files

Master files are text files that contain RRs in text form. Since the contents of a zone can be expressed in the form of a list of RRs a master file is most often used to define a zone, though it can be used to list a cache's contents. Hence, this section first discusses the format of RRs in a master file, and then the special considerations when a master file is used to create a zone in some name server.

RFC-1034/5 Domains: Name Space and Resource Records

Master File Format

The format of these files is a sequence of entries. Entries are predominantly line-oriented, though parentheses can be used to continue a list of items across a line boundary, and text literals can contain CRLF within the text. Any combination of tabs and spaces act as a delimiter between the separate items that make up an entry. The end of any line in the master file can end with a comment. The comment starts with a ";" (semicolon).

The following entries are defined:

```
<blank>[<comment>]

$ORIGIN <domain-name> [<comment>]

$INCLUDE <file-name> [<domain-name>] [<comment>]

<domain-name><rr> [<comment>]

<blank><rr> [<comment>]
```

Blank lines, with or without comments, are allowed anywhere in the file.

Two control entries are defined: $ORIGIN and $INCLUDE. $ORIGIN is followed by a domain name, and resets the current origin for relative domain names to the stated name. $INCLUDE inserts the named file into the current file, and may optionally specify a domain name that sets the relative domain name origin for the included file. $INCLUDE may also have a comment. Note that a $INCLUDE entry never changes the relative origin of the parent file, regardless of changes to the relative origin made within the included file.

The last two forms represent RRs. If an entry for an RR begins with a blank, then the RR is assumed to be owned by the last stated owner. If an RR entry begins with a <domain-name>, then the owner name is reset.

<rr> contents take one of the following forms:

```
[<TTL>] [<class>] <type> <RDATA>

[<class>] [<TTL>] <type> <RDATA>
```

The RR begins with optional TTL and class fields, followed by a type and RDATA field appropriate to the type and class. Class and type use the standard mnemonics, TTL is a decimal integer. Omitted class and TTL values are default to the last explicitly stated values. Since type and class mnemonics are disjoint, the parse is unique. (Note that this order is different from the order used in examples and the order used in the actual RRs; the given order allows easier parsing and defaulting.)

<domain-name>s make up a large share of the data in the master file. The labels in the domain name are expressed as character strings and separated by dots. Quoting conventions allow arbitrary characters to be stored in domain names. Domain names that end in a dot are called absolute, and are taken as complete. Domain names which do not end in a dot are called relative; the actual domain name is the concatenation of the relative part with an origin specified in a $ORIGIN, $INCLUDE, or as an argument to the master file loading routine. A relative name is an error when no origin is available.

<character-string> is expressed in one or two ways: as a contiguous set of characters without interior spaces, or as a string beginning with a " and ending with a ". Inside a " delimited string any character can occur, except for a " itself, which must be quoted using \ (back slash).

Because these files are text files several special encodings are necessary to allow arbitrary data to be loaded. In particular:

	of the root.
@	A free standing @ is used to denote the current origin.
\X	where X is any character other than a digit (0-9), is used to quote that character so that its special meaning does not apply. For example, "\." can be used to place a dot character in a label.
\DDD	where each D is a digit is the octet corresponding to the decimal number described by DDD. The resulting octet is assumed to be text and is not checked for special meaning.
( )	Parentheses are used to group data that crosses a line boundary. In effect, line terminations are not recognized within parentheses.
;	Semicolon is used to start a comment; the remainder of the line is ignored.

Use of Master Files to Define Zones

When a master file is used to load a zone, the operation should be suppressed if any errors are encountered in the master file. The rationale for this is that a single error can have widespread consequences. For example, suppose that the RRs defining a delegation have syntax errors; then the server will return authoritative name errors for all names in the subzone (except in the case where the subzone is also present on the server).

Several other validity checks that should be performed in addition to insuring that the file is syntactically correct:

1. All RRs in the file should have the same class.

2. Exactly one SOA RR should be present at the top of the zone.

3. If delegations are present and glue information is required, it should be present.

4. Information present outside of the authoritative nodes in the zone should be glue information, rather than the result of an origin or similar error.

Master File Example

The following is an example file which might be used to define the `ISI.EDU` zone.and is loaded with an origin of `ISI.EDU`:

```
@   IN  SOA     VENERA      Action\.domains (
                                 20     ; SERIAL
                                 7200   ; REFRESH
                                 600    ; RETRY
                                 3600000; EXPIRE
                                 60)    ; MINIMUM

        NS      A.ISI.EDU.
        NS      VENERA
        NS      VAXA
        MX      10      VENERA
        MX      20      VAXA

A       A       26.3.0.103

VENERA  A       10.1.0.52
        A       128.9.0.32

VAXA    A       10.2.0.27
        A       128.9.0.33


$INCLUDE <SUBSYS>ISI-MAILBOXES.TXT
```

Where the file `<SUBSYS>ISI-MAILBOXES.TXT` is:

```
    MOE     MB      A.ISI.EDU.
    LARRY   MB      A.ISI.EDU.
```

Aliases and Canonical Names

In existing systems, hosts and other resources often have several names that identify the same resource. For example, the names `C.ISI.EDU` and `USC-ISIC.ARPA` both identify the same host. Similarly, in the case of mailboxes, many organizations provide many names that actually go to the same mailbox; for example `Mockapetris@C.ISI.EDU`, `Mockapetris@B.ISI.EDU`, and `PVM@ISI.EDU` all go to the same mailbox (although the mechanism behind this is somewhat complicated).

Most of these systems have a notion that one of the equivalent set of names is the canonical or primary name and all others are aliases.

The domain system provides such a feature using the canonical name (CNAME) RR. A CNAME RR identifies its owner name as an alias, and specifies the corresponding canonical name in the RDATA section of the RR. If a CNAME RR is present at a node, no other data should be present; this ensures that the data for a canonical name and its aliases cannot be different. This rule also insures that a cached CNAME can be used without checking with an authoritative server for other RR types.

CNAME RRs cause special action in DNS software. When a name server fails to find a desired RR in the resource set associated with the domain name, it checks to see if the resource set consists of a CNAME record with a matching class. If so, the name server includes the CNAME record in the response and restarts the query at the domain name specified in the data field of the CNAME record. The one exception to this rule is that queries which match the CNAME type are not restarted.

For example, suppose a name server was processing a query with for `USC- ISIC.ARPA`, asking for type A information, and had the following resource records:

```
USC-ISIC.ARPA   IN      CNAME   C.ISI.EDU

C.ISI.EDU       IN      A       10.0.0.52
```

Both of these RRs would be returned in the response to the type A query, while a type CNAME or * query should return just the CNAME.

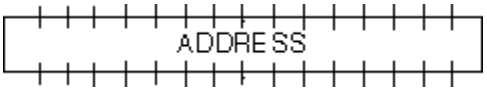
Domain names in RRs which point at another name should always point at the primary name and not the alias. This avoids extra indirections in accessing information. For example, the address to name RR for the above host should be:

```
52.0.0.10.IN-ADDR.ARPA  IN      PTR     C.ISI.EDU
```

rather than pointing at `USC-ISIC.ARPA`. Of course, by the robustness principle, domain software should not fail when presented with CNAME chains or loops; CNAME chains should be followed and CNAME loops signalled as an error.

ARPA Internet Specific RRs

A RDATA Format



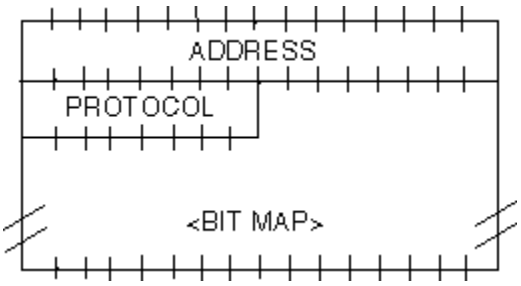
where:

ADDRESS A 32 bit Internet address.

Hosts that have multiple Internet addresses will have multiple A records.

A records cause no additional section processing. The RDATA section of an A line in a master file is an Internet address expressed as four decimal numbers separated by dots without any imbedded spaces (e.g., "`10.2.0.52`" or "`192.0.5.6`").

WKS RDATA Format



where:

ADDRESS An 32 bit Internet address

PROTOCOL An 8 bit IP protocol number

<BIT MAP> A variable length bit map. The bit map must be a multiple of 8 bits long.

The WKS record is used to describe the well known services supported by a particular protocol on a particular internet address. The PROTOCOL field specifies an IP protocol number, and the bit map has one bit per port of the specified protocol. The first bit corresponds to port 0, the second to port 1, etc. If the bit map does not include a bit for a protocol of interest, that bit is assumed zero. The appropriate values and mnemonics for ports and protocols are specified in RFC-1010.

For example, if PROTOCOL=TCP (6), the 26th bit corresponds to TCP port 25 (SMTP). If this bit is set, a SMTP server should be listening on TCP port 25; if zero, SMTP service is not supported on the specified address.

The purpose of WKS RRs is to provide availability information for servers for TCP and UDP. If a server supports both TCP and UDP, or has multiple Internet addresses, then multiple WKS RRs are used.

WKS RRs cause no additional section processing.

In master files, both ports and protocols are expressed using mnemonics or decimal numbers.

IN-ADDR.ARPA Domain

The Internet uses a special domain to support gateway location and Internet address to host mapping. Other classes may employ a similar strategy in other domains. The intent of this domain is to provide a guaranteed method to perform host address to host name mapping, and to facilitate queries to locate all gateways on a particular network in the Internet.

Note that both of these services are similar to functions that could be performed by inverse queries; the difference is that this part of the domain name space is structured according to address, and hence can guarantee that the appropriate data can be located without an exhaustive search of the domain space.

The domain begins at `IN-ADDR.ARPA` and has a substructure which follows the Internet addressing structure.

Domain names in the `IN-ADDR.ARPA` domain are defined to have up to four labels in addition to the `IN-ADDR.ARPA` suffix. Each label represents one octet of an Internet address, and is expressed as a character string for a decimal value in the range 0-255 (with leading zeros omitted except in the case of a zero octet which is represented by a single zero).

Host addresses are represented by domain names that have all four labels specified. Thus data for Internet address `10.2.0.52` is located at domain name `52.0.2.10.IN-ADDR.ARPA`. The reversal, though awkward to read, allows zones to be delegated which are exactly one network of address space. For example, `10.IN-ADDR.ARPA` can be a zone containing data for the ARPANET, while `26.IN-ADDR.ARPA` can be a separate zone for MILNET. Address nodes are used to hold pointers to primary host names in the normal domain space.

Network numbers correspond to some non-terminal nodes at various depths in the `IN-ADDR.ARPA` domain, since Internet network numbers are either 1, 2, or 3 octets. Network nodes are used to hold pointers to the primary host names of gateways attached to that network. Since a gateway is, by definition, on more than one network, it will typically have two or more network nodes which point at it. Gateways will also have host level pointers at their fully qualified addresses.

Both the gateway pointers at network nodes and the normal host pointers at full address nodes use the PTR RR to point back to the primary domain names of the corresponding hosts.

For example, the `IN-ADDR.ARPA` domain will contain information about the ISI gateway between net 10 and 26, an MIT gateway from net 10 to MIT's net 18, and hosts `A.ISI.EDU` and `MULTICS.MIT.EDU`. Assuming that ISI gateway has addresses `10.2.0.22` and `26.0.0.103`, and a name MILNET-`GW.ISI.EDU`, and the MIT gateway has addresses `10.0.0.77` and `18.10.0.4` and a name `GW.LCS.MIT.EDU`, the domain database would contain:

```
10.IN-ADDR.ARPA.            PTR MILNET-GW.ISI.EDU.
10.IN-ADDR.ARPA.            PTR GW.LCS.MIT.EDU.
18.IN-ADDR.ARPA.            PTR GW.LCS.MIT.EDU.
26.IN-ADDR.ARPA.            PTR MILNET-GW.ISI.EDU.
22.0.2.10.IN-ADDR.ARPA.     PTR MILNET-GW.ISI.EDU.
103.0.0.26.IN-ADDR.ARPA.    PTR MILNET-GW.ISI.EDU.
77.0.0.10.IN-ADDR.ARPA.     PTR GW.LCS.MIT.EDU.
```

```
4.0.10.18.IN-ADDR.ARPA.     PTR GW.LCS.MIT.EDU.
103.0.3.26.IN-ADDR.ARPA.    PTR A.ISI.EDU.
6.0.0.10.IN-ADDR.ARPA.      PTR MULTICS.MIT.EDU.
```

Thus a program which wanted to locate gateways on net 10 would originate a query of the form `QTYPE=PTR, QCLASS=IN, QNAME=10.IN-ADDR.ARPA`. It would receive two RRs in response:

```
10.IN-ADDR.ARPA.            PTR MILNET-GW.ISI.EDU.
10.IN-ADDR.ARPA.            PTR GW.LCS.MIT.EDU.
```

The program could then originate `QTYPE=A, QCLASS=IN` queries for MILNET-`GW.ISI.EDU.` and `GW.LCS.MIT.EDU.` to discover the Internet addresses of these gateways.

A resolver which wanted to find the host name corresponding to Internet host address **`10.0.0.6`** would pursue a query of the form `QTYPE=PTR, QCLASS=IN, QNAME=6.0.0.10.IN-ADDR.ARPA`, and would receive:

```
6.0.0.10.IN-ADDR.ARPA.      PTR MULTICS.MIT.EDU.
```

Several cautions apply to the use of these services:

- Since the `IN-ADDR.ARPA` special domain and the normal domain for a particular host or gateway will be in different zones, the possibility exists that that the data may be inconsistent.
- Gateways will often have two names in separate domains, only one of which can be primary.
- Systems that use the domain database to initialize their routing tables must start with enough gateway information to guarantee that they can access the appropriate name server.
- The gateway data only reflects the existence of a gateway in a manner equivalent to the current `HOSTS.TXT` file. It doesn't replace the dynamic availability information from GGP or EGP.

Defining New Types, Classes, and Special Name Spaces

The previously defined types and classes are the ones in use as of the date of this memo. New definitions should be expected. This section makes some recommendations to designers considering additions to the existing facilities. The mailing list NAMEDROPPERS@SRI-NIC.ARPA is the forum where general discussion of design issues takes place.

In general, a new type is appropriate when new information is to be added to the database about an existing object, or we need new data formats for some totally new object. Designers should attempt to define types and their RDATA formats that are generally applicable to all classes, and which avoid duplication of information. New classes are appropriate when the DNS is to be used for a new protocol, etc which requires new class-specific data formats, or when a copy of the existing name space is desired, but a separate management domain is necessary.

New types and classes need mnemonics for master files; the format of the master files requires that the mnemonics for type and class be disjoint.

TYPE and CLASS values must be a proper subset of QTYPEs and QCLASSes respectively.

The present system uses multiple RRs to represent multiple values of a type rather than storing multiple values in the RDATA section of a single RR. This is less efficient for most applications, but does keep RRs shorter. The multiple RRs assumption is incorporated in some experimental work on dynamic update methods.

The present system attempts to minimize the duplication of data in the database in order to insure consistency. Thus, in order to find the address of the host for a mail exchange, you map the mail domain name to a host name, then the host name to addresses, rather than a direct mapping to host address. This approach is preferred because it avoids the opportunity for inconsistency.

In defining a new type of data, multiple RR types should not be used to create an ordering between entries or express different formats for equivalent bindings, instead this information should be carried in the body of the RR and a single type used. This policy avoids problems with caching multiple types and defining QTYPEs to match multiple types.

For example, the original form of mail exchange binding used two RR types one to represent a "closer" exchange (MD) and one to represent a "less close" exchange (MF). The difficulty is that the presence of one RR type in a cache doesn't convey any information about the other because the query which acquired the cached information might have used a QTYPE of MF, MD, or MAILA (which matched both). The redesigned service used a single type (MX) with a "preference" value in the RDATA section which can order different RRs. However, if any MX RRs are found in the cache, then all should be there.

RFC-1034/5 Domains: Name Space and Resource Records

Queries

Queries are messages which may be sent to a name server to provoke a response. In the Internet, queries are carried in UDP datagrams or over TCP connections. The response by the name server either answers the question posed in the query, refers the requester to another set of name servers, or signals some error condition.

In general, the user does not generate queries directly, but instead makes a request to a resolver which in turn sends one or more queries to name servers and deals with the error conditions and referrals that may result. Of course, the possible questions which can be asked in a query does shape the kind of service a resolver can provide.

DNS queries and responses are carried in a standard message format. The message format has a header containing a number of fixed fields which are always present, and four sections which carry query parameters and RRs.

The most important field in the header is a four bit field called an opcode which separates different queries. Of the possible 16 values, one (standard query)is part of the official protocol, two (inverse query and status query) are options, one (completion) is obsolete, and the rest are unassigned.

The four sections are:

Question Carries the query name and other query parameters.

Answer Carries RRs which directly answer the query.

Authority Carries RRs which describe other authoritative servers. May optionally carry the SOA RR for the authoritative data in the answer section.

Additional Carries RRs which may be helpful in using the RRs in the other sections.

Note that the content, but not the format, of these sections varies with header opcode.

Standard Queries

A standard query specifies a target domain name (QNAME), query type (QTYPE), and query class (QCLASS) and asks for RRs which match. This type of query makes up such a vast majority of DNS queries that we use the term "query" to mean standard query unless otherwise specified. The QTYPE and QCLASS fields are each 16 bits long, and are a superset of defined types and classes.

The QTYPE field may contain:

<any type> matches just that type. (e.g., A, PTR).

AXFR special zone transfer QTYPE.

MAILB matches all mail box related RRs (e.g. MB and MG).

***** matches all RR types.

The QCLASS field may contain:

<any class> matches just that class (e.g., IN, CH).

***** matches aLL RR classes.

Using the query domain name, QTYPE, and QCLASS, the name server looks for matching RRs. In addition to relevant records, the name server may return RRs that point toward a name server that has the desired information or RRs that are expected to be useful in interpreting the relevant RRs. For example, a name server that doesn't have the requested information may know a name server that does; a name server that returns a domain name in a relevant RR may also return the RR that binds that domain name to an address.

For example, a mailer tying to send mail to `Mockapetris@ISI.EDU` might ask the resolver for mail information about `ISI.EDU`, resulting in a query for `QNAME=ISI.EDU`, `QTYPE=MX`, `QCLASS=IN`. The response's answer section would be:

```
ISI.EDU.        MX      10 VENERA.ISI.EDU.
                MX      10 VAXA.ISI.EDU.
```

while the additional section might be:

```
VAXA.ISI.EDU.   A       10.2.0.27
                A       128.9.0.33
VENERA.ISI.EDU. A       10.1.0.52
                A       128.9.0.32
```

Because the server assumes that if the requester wants mail exchange information, it will probably want the addresses of the mail exchanges soon afterward.

Note that the QCLASS=* construct requires special interpretation regarding authority. Since a particular name server may not know all of the classes availablein the domain system, it can never know if it is authoritative for all classes. Hence responses to QCLASS=* queries can never be authoritative.

For a further discussion of standard queries, see Standard Query Processing under Name Server Implementation.

Inverse Queries (Optional)

Name servers may also support inverse queries that map a particular resource to a domain name or domain names that have that resource. For example, while a standard query might map a domain name to a SOA RR, the corresponding inverse query might map the SOA RR back to the domain name.

Implementation of this service is optional in a name server, but all name servers must at least be able to understand an inverse query message and return a not-implemented error response.

The domain system cannot guarantee the completeness or uniqueness of inverse queries because the domain system is organized by domain name rather than by host address or any other resource type. Inverse queries are primarily useful for debugging and database maintenance activities.

Inverse queries may not return the proper TTL, and do not indicate cases where the identified RR is one of a set (for example, one address for a host having multiple addresses). Therefore, the RRs returned in inverse queries should never be cached.

Inverse queries are NOT an acceptable method for mapping host addresses to host names; use the `IN-ADDR.ARPA` domain instead.

For a further discussion of inverse queries, see Inverse Queries under Name Server Implementation.

Status Queries (Experimental)

To be defined.

Completion Queries (Obsolete)

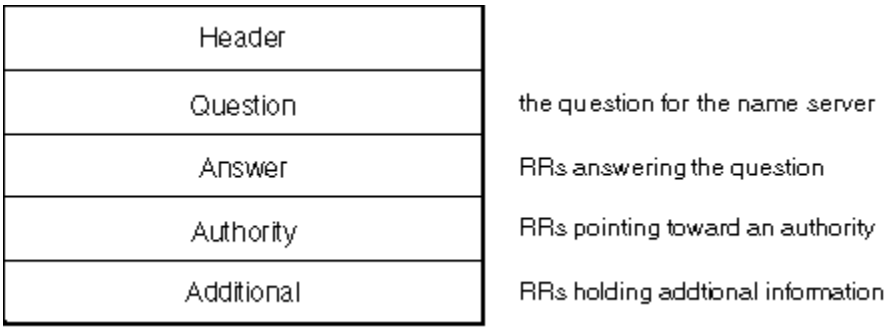
The optional completion services described in RFCs 882 and RFC-883 have been deleted. Redesigned services may become available in the future, or the opcodes may be reclaimed for other use.

Messages

RFC-1034/5 Domain Name System: Messages

Format

All communications inside of the domain protocol are carried in a single format called a message. The top level format of message is divided into 5 sections (some of which are empty in certain cases) shown below:



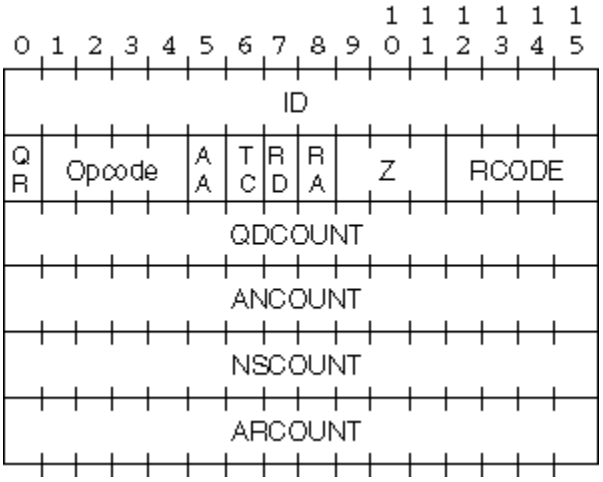
The header section is always present. The header includes fields that specify which of the remaining sections are present, and also specify whether the message is a query or a response, a standard query or some other opcode, etc.

The names of the sections after the header are derived from their use in standard queries. The question section contains fields that describe a question to a name server. These fields are a query type (QTYPE), a query class (QCLASS), and a query domain name (QNAME). The last three sections have the same format: a possibly empty list of concatenated resource records (RRs). The answer section contains RRs that answer the question; the authority section contains RRs that point toward an authoritative name server; the additional records section contains RRs which relate to the query, but are not strictly answers for the question.

RFC-1034/5 Domain Name System: Messages

Header Section Format

The header contains the following fields:



where:

ID A 16 bit identifier assigned by the program that generates any kind of query. This identifier is copied the corresponding reply and can be used by the requester to match up replies to outstanding queries.

QR A one bit field that specifies whether this message is a query (0), or a response (1).

OPCODE A four bit field that specifies kind of query in this message. This value is set by the originator of a query and copied into the response. The values are:

- **0** a standard query (QUERY)
- **1** an inverse query (IQUERY)
- **2** a server status request (STATUS)
- **3-15** reserved for future use

AA Authoritative Answer - this bit is valid in responses, and specifies that the responding name server is an authority for the domain name in question section.

Note that the contents of the answer section may have multiple owner names because of aliases. The AA bit corresponds to the name which matches the query name, or the first owner name in the answer section.

TC TrunCation - specifies that this message was truncated due to length greater than that permitted on the transmission channel.

RD Recursion Desired - this bit may be set in a query and is copied into the response. If RD is set, it directs the name server to pursue the query recursively. Recursive query support is optional.

RA Recursion Available - this be is set or cleared in a response, and denotes whether recursive query support is available in the name server.

Z Reserved for future use. Must be zero in all queries and responses.

RCODE Response code - this 4 bit field is set as part of responses. The values have the following interpretation:

- **0** No error condition
- **1** Format error - The name server was unable to interpret the query.
- **2** Server failure - The name server was unable to process this query due to a problem with the name server.
- **3** Name Error - Meaningful only for responses from an authoritative name server, this code signifies that the domain name referenced in the query does not exist.
- **4** Not Implemented - The name server does not support the requested kind of query.
- **5** Refused - The name server refuses to perform the specified operation for policy reasons. For example, a name server may not wish to provide the information to the particular requester, or a name server may not wish to perform a particular operation (e.g., zone transfer) for particular data.
- **6-15** Reserved for future use.

QDCOUNT an unsigned 16 bit integer specifying the number of entries in the question section.

ANCOUNT an unsigned 16 bit integer specifying the number of resource records in the answer section.

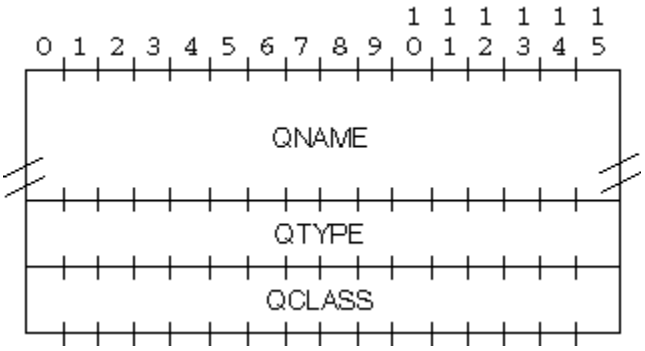
NSCOUNT an unsigned 16 bit integer specifying the number of name records in the authority records section.

ARCOUNT an unsigned 16 bit integer specifying the number of resource records in the additional records section.

RFC-1034/5 Domain Name System: Messages

Question Section Format

The question section is used to carry the "question" in most queries, i.e., the parameters that define what is being asked. The section contains QDCOUNT (usually 1) entries, each of the following format:



where:

QNAME a domain name represented as a sequence of labels, where each label consists of a length octet followed by that number of octets. The domain name terminates with the zero length octet for the null label of the root. Note that this field may be an odd number of octets; no padding is used.

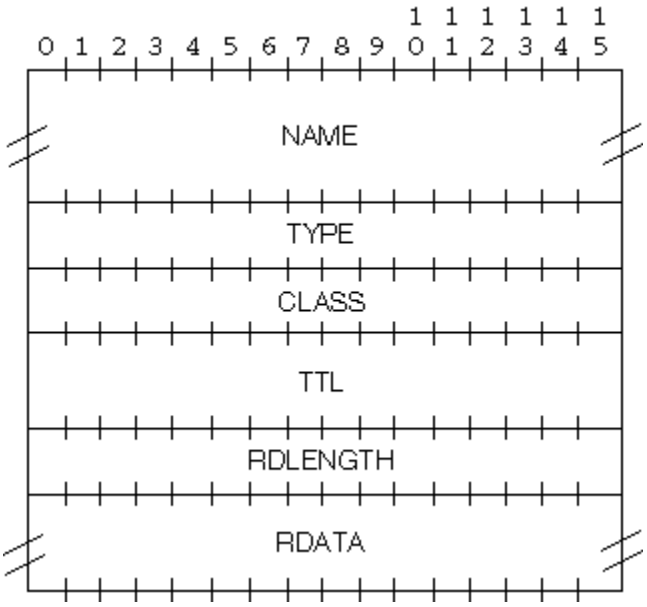
QTYPE a two octet code which specifies the type of the query. The values for this field include all codes valid for a TYPE field, together with some more general codes which can match more than one type of RR.

QCLASS a two octet code that specifies the class of the query. For example, the QCLASS field is IN for the Internet.

RFC-1034/5 Domain Name System: Messages

Resource Record Format

The answer, authority, and additional sections all share the same format: a variable number of resource records, where the number of records is specified in the corresponding count field in the header. Each resource record has the following format:



where:

NAME a domain name to which this resource record pertains.

TYPE two octets containing one of the RR type codes. This field specifies the meaning of the data in the RDATA field.

CLASS two octets which specify the class of the data in the RDATA field.

TTL a 32 bit unsigned integer that specifies the time interval (in seconds) that the resource record may be cached before it should be discarded. Zero values are interpreted to mean that the RR can only be used for the transaction in progress, and should not be cached.

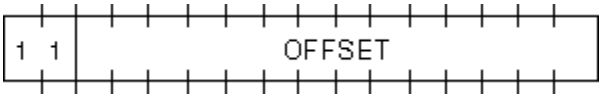
RDLENGTH an unsigned 16 bit integer that specifies the length in octets of the RDATA field.

RDATA a variable length string of octets that describes the resource. The format of this information varies according to the TYPE and CLASS of the resource record. For example, the if the TYPE is A and the CLASS is IN, the RDATA field is a 4 octet ARPA Internet address.

Message Compression

In order to reduce the size of messages, the domain system utilizes a compression scheme which eliminates the repetition of domain names in a message. In this scheme, an entire domain name or a list of labels at the end of a domain name is replaced with a pointer to a prior occurance of the same name.

The pointer takes the form of a two octet sequence:



The first two bits are ones. This allows a pointer to be distinguished from a label, since the label must begin with two zero bits because labels are restricted to 63 octets or less. (The 10 and 01 combinations are reserved for future use.) The OFFSET field specifies an offset from the start of the message (i.e., the first octet of the ID field in the domain header). A zero offset specifies the first byte of the ID field, etc.

The compression scheme allows a domain name in a message to be represented as either:

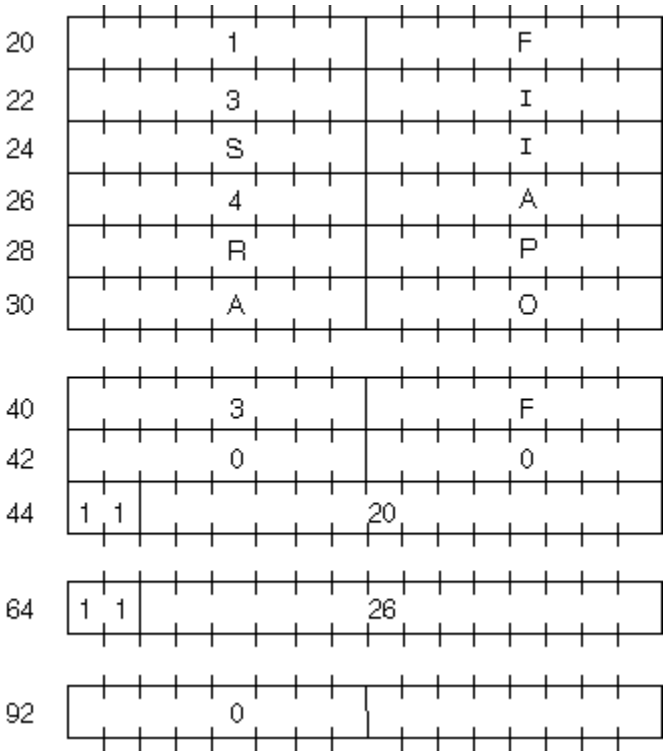
- a sequence of labels ending in a zero octet
- a pointer
- a sequence of labels ending with a pointer

Pointers can only be used for occurances of a domain name where the format is not class specific. If this were not the case, a name server or resolver would be required to know the format of all RRs it handled. As yet, there are no such cases, but they may occur in future RDATA formats.

If a domain name is contained in a part of the message subject to a length field (such as the RDATA section of an RR), and compression is used, the length of the compressed name is used in the length calculation, rather than the length of the expanded name.

Programs are free to avoid using pointers in messages they generate, although this will reduce datagram capacity, and may cause truncation. However all programs are required to understand arriving messages that contain pointers.

For example, a datagram might need to use the domain names `F.ISI.ARPA`, `FOO.F.ISI.ARPA`, `ARPA`, and the root. Ignoring the other fields of the message, these domain names might be represented as:



The domain name for `F.ISI.ARPA` is shown at offset 20. The domain name `FOO.F.ISI.ARPA` is shown at offset 40; this definition uses a pointer to concatenate a label for FOO to the previously defined `F.ISI.ARPA`. The domain name ARPA is defined at offset 64 using a pointer to the ARPA component of the name `F.ISI.ARPA` at 20; note that this pointer relies on ARPA being the last label in the string at 20. The root domain name is defined by a single octet of zeros at 92; the root domain name has no labels.

RFC-1034/5 Domain Name System: Messages

Transport

The DNS assumes that messages will be transmitted as datagrams or in a byte stream carried by a virtual circuit. While virtual circuits can be used for any DNS activity, datagrams are preferred for queries due to their lower overhead and better performance. Zone refresh activities must use virtual circuits because of the need for reliable transfer.

The Internet supports name server access using TCP (RFC-793) on server port 53 (decimal) as well as datagram access using UDP (RFC-768) on UDP port 53 (decimal).

UDP usage

Messages sent using UDP user server port 53 (decimal).

Messages carried by UDP are restricted to 512 bytes (not counting the IP or UDP headers). Longer messages are truncated and the TC bit is set in the header.

UDP is not acceptable for zone transfers, but is the recommended method for standard queries in the Internet. Queries sent using UDP may be lost, and hence a retransmission strategy is required. Queries or their responses may be reordered by the network, or by processing in name servers, so resolvers should not depend on them being returned in order.

The optimal UDP retransmission policy will vary with performance of the Internet and the needs of the client, but the following are recommended:

- The client should try other servers and server addresses before repeating a query to a specific address of a server.

- The retransmission interval should be based on prior statistics if possible. Too aggressive retransmission can easily slow responses for the community at large. Depending on how well connected the client is to its expected servers, the minimum retransmission interval should be 2-5 seconds.

More suggestions on server selection and retransmission policy can be found in the resolver section of this memo.

TCP usage

Messages sent over TCP connections use server port 53 (decimal). The message is prefixed with a two byte length field which gives the message length, excluding the two byte length field. This length field allows the low-level processing to assemble a complete message before beginning to parse it.

Several connection management policies are recommended:

- The server should not block other activities waiting for TCP data.

- The server should support multiple connections.

- The server should assume that the client will initiate connection

closing, and should delay closing its end of the connection until all outstanding client requests have been satisfied.

- If the server needs to close a dormant connection to reclaim resources, it should wait until the connection has been idle for a period on the order of two minutes. In particular, the server should allow the SOA and AXFR request sequence (which begins a refresh operation) to be made on a single connection. Since the server would be unable to answer queries anyway, a unilateral close or reset may be used instead of a graceful close.

RFC-1034/5 Domain Name System

Name Servers

RFC-1034/5 Domain System: Name Servers

Introduction

Name servers are the repositories of information that make up the domain database. The database is divided up into sections called zones, which are distributed among the name servers. While name servers can have several optional functions and sources of data, the essential task of a name server is to answer queriesusing data in its zones. By design, name servers can answer queries in a simple manner; the response can always be generated using only local data, and eithercontains the answer to the question or a referral to other name servers "closer" to the desired information.

A given zone will be available from several name servers to insure its availability in spite of host or communication link failure. By administrative fiat, werequire every zone to be available on at least two servers, and many zones have more redundancy than that.

A given name server will typically support one or more zones, but this gives it authoritative information about only a small section of the domain tree. It may also have some cached non-authoritative data about other parts of the tree. The name server marks its responses to queries so that the requester can tell whether the response comes from authoritative data or not.

Architecture

The optimal structure for the name server will depend on the host operating system and whether the name server is integrated with resolver operations, either by supporting recursive service, or by sharing its database with a resolver. This section discusses implementation considerations for a name server which shares a database with a resolver, but most of these concerns are present in any name server.

Control

A name server must employ multiple concurrent activities, whether they are implemented as separate tasks in the host's OS or multiplexing inside a single name server program. It is simply not acceptable for a name server to block the service of UDP requests while it waits for TCP data for refreshing or query activities. Similarly, a name server should not attempt to provide recursive service without processing such requests in parallel, though it may choose to serialize requests from a single client, or to regard identical requests from the same client as duplicates. A name server should not substantially delay requests while it reloads a zone from master files or while it incorporates a newly refreshed zone into its database.

Database

While name server implementations are free to use any internal data structures they choose, the suggested structure consists of three major parts:

- A "catalog" data structure which lists the zones available to this server, and a "pointer" to the zone data structure. The main purpose of this structure is to find the nearest ancestor zone, if any, for arriving standard queries.

- Separate data structures for each of the zones held by the name server.

- A data structure for cached data. (or perhaps separate caches for different classes)

All of these data structures can be implemented an identical tree structure format, with different data chained off the nodes in different parts: in the catalog the data is pointers to zones, while in the zone and cache data structures, the data will be RRs. In designing the tree framework the designer should recognize that query processing will need to traverse the tree using case-insensitive label comparisons; and that in real data, a few nodes have a very high branching factor (100-1000 or more), but the vast majority have a very low branching factor (0-1).

One way to solve the case problem is to store the labels for each node in two pieces: a standardized-case representation of the label where all ASCII characters are in a single case, together with a bit mask that denotes which characters are actually of a different case. The branching factor diversity can be handled using a simple linked list for a node until the branching factor exceeds some threshold, and transitioning to a hash structure after the threshold is exceeded. In any case, hash structures used to store tree sections must insure that hash functions and procedures preserve the casing conventions of the DNS.

The use of separate structures for the different parts of the database is motivated by several factors:

- The catalog structure can be an almost static structure that need change only when the system administrator changes the zones supported by the server. This structure can also be used to store parameters used to control refreshing activities.

- The individual data structures for zones allow a zone to be replaced simply by changing a pointer in the catalog. Zone refresh operations can build a new structure and, when complete, splice it into the database via a simple pointer replacement. It is very important that when a zone is refreshed, queries should not use old and new data simultaneously.

- With the proper search procedures, authoritative data in zones will always "hide", and hence take precedence over, cached data.

- Errors in zone definitions that cause overlapping zones, etc., may cause erroneous responses to queries, but problem determination is simplified, and the contents of one "bad" zone can't corrupt another.

- Since the cache is most frequently updated, it is most vulnerable to corruption during system restarts. It can also become full of expired RR data. In either case, it can easily be discarded without disturbing zone data.

A major aspect of database design is selecting a structure which allows the name server to deal with crashes of the name server's host. State information which a name server should save across system crashes includes the catalog structure (including the state of refreshing for each zone) and the zone data itself.

Time

Both the TTL data for RRs and the timing data for refreshing activities depends on 32 bit timers in units of seconds. Inside the database, refresh timers and TTLs for cached data conceptually "count down", while data in the zone stays with constant TTLs.

A recommended implementation strategy is to store time in two ways: as a relative increment and as an absolute time. One way to do this is to use positive 32 bit numbers for one type and negative numbers for the other. The RRs in zones use relative times; the refresh timers and cache data use absolute times. Absolute numbers are taken with respect to some known origin and converted to relative values when placed in the response to a query. When an absolute TTL is negative after conversion to relative, then the data is expired and should be ignored.

RFC-1034/5 Domain System: Name Servers

How the Database is Dvided into Zones

The domain database is partitioned in two ways: by class, and by "cuts" made in the name space between nodes.

The class partition is simple. The database for any class is organized, delegated, and maintained separately from all other classes. Since, by convention, the name spaces are the same for all classes, the separate classes can be thought of as an array of parallel namespace trees. Note that the data attached to nodes will be different for these different parallel classes. The most common reasons for creating a new class are the necessity for a new data format for existing types or a desire for a separately managed version of the existing name space.

Within a class, "cuts" in the name space can be made between any two adjacent nodes. After all cuts are made, each group of connected name space is a separatezone. The zone is said to be authoritative for all names in the connected region. Note that the "cuts" in the name space may be in different places for different classes, the name servers may be different, etc.

These rules mean that every zone has at least one node, and hence domain name, for which it is authoritative, and all of the nodes in a particular zone are connected. Given, the tree structure, every zone has a highest node which is closer to the root than any other node in the zone. The name of this node is often used to identify the zone.

It would be possible, though not particularly useful, to partition the name space so that each domain name was in a separate zone or so that all nodes were in a single zone. Instead, the database is partitioned at points where a particular organization wants to take over control of a subtree. Once an organization controls its own zone it can unilaterally change the data in the zone, grow new tree sections connected to the zone, delete existing nodes, or delegate new subzones under its zone.

If the organization has substructure, it may want to make further internal partitions to achieve nested delegations of name space control. In some cases, suchdivisions are made purely to make database maintenance more convenient.

RFC-1034/5 Domain System: Name Servers

Technical Considerations

The data that describes a zone has four major parts:

- Authoritative data for all nodes within the zone.
- Data that defines the top node of the zone (can be thought of as part of the authoritative data).
- Data that describes delegated subzones, i.e., cuts around the bottom of the zone.
- Data that allows access to name servers for subzones (sometimes called "glue" data).

All of this data is expressed in the form of RRs, so a zone can be completely described in terms of a set of RRs. Whole zones can be transferred between name servers by transferring the RRs, either carried in a series of messages or by FTPing a master file which is a textual representation.

The authoritative data for a zone is simply all of the RRs attached to all of the nodes from the top node of the zone down to leaf nodes or nodes above cuts around the bottom edge of the zone.

Though logically part of the authoritative data, the RRs that describe the top node of the zone are especially important to the zone's management. These RRs are of two types: name server RRs that list, one per RR, all of the servers for the zone, and a single SOA RR that describes zone management parameters.

The RRs that describe cuts around the bottom of the zone are NS RRs that name the servers for the subzones. Since the cuts are between nodes, these RRs are NOT part of the authoritative data of the zone, and should be exactly the same as the corresponding RRs in the top node of the subzone. Since name servers are always associated with zone boundaries, NS RRs are only found at nodes which are the top node of some zone. In the data that makes up a zone, NS RRs are found at the top node of the zone (and are authoritative) and at cuts around the bottom of the zone (where they are not authoritative), but never in between.

One of the goals of the zone structure is that any zone have all the data required to set up communications with the name servers for any subzones. That is, parent zones have all the information needed to access servers for their children zones. The NS RRs that name the servers for subzones are often not enough forthis task since they name the servers, but do not give their addresses. In particular, if the name of the name server is itself in the subzone, we could be faced with the situation where the NS RRs tell us that in order to learn a name server's address, we should contact the server using the address we wish to learn. To fix this problem, a zone contains "glue" RRs which are not part of the authoritative data, and are address RRs for the servers. These RRs are only necessary if the name server's name is "below" the cut, and are only used as part of a referral response.

Administrative Considerations

When some organization wants to control its own domain, the first step is to identify the proper parent zone, and get the parent zone's owners to agree to the delegation of control. While there are no particular technical constraints dealing with where in the tree this can be done, there are some administrative groupings discussed in RFC-1032 which deal with top level organization, and middle level zones are free to create their own rules. For example, one university might choose to use a single zone, while another might choose to organize by subzones dedicated to individual departments or schools. RFC-1033 catalogs available DNS software an discusses administration procedures.

Once the proper name for the new subzone is selected, the new owners should be required to demonstrate redundant name server support. Note that there is no requirement that the servers for a zone reside in a host which has a name in that domain. In many cases, a zone will be more accessible to the internet at largeif its servers are widely distributed rather than being within the physical facilities controlled by the same organization that manages the zone. For example,in the current DNS, one of the name servers for the United Kingdom, or UK domain, is found in the US. This allows US hosts to get UK data without using limited transatlantic bandwidth.

As the last installation step, the delegation NS RRs and glue RRs necessary to make the delegation effective should be added to the parent zone. The administrators of both zones should insure that the NS and glue RRs which mark both sides of the cut are consistent and remain so.

RFC-1034/5 Domain System: Name Servers

Queries and Responses

The principal activity of name servers is to answer standard queries. Both the query and its response are carried in a standard message format (RFC_974). The query contains a QTYPE, QCLASS, and QNAME, which describe the types and classes of desired information and the name of interest.

The way that the name server answers the query depends upon whether it is operating in recursive mode or not:

- The simplest mode for the server is non-recursive, since it can answer queries using only local information: the response contains an error, the answer, or a referral to some other server "closer" to the answer. All name servers must implement non-recursive queries.

- The simplest mode for the client is recursive, since in this mode the name server acts in the role of a resolver and returns either an error or the answer, but never referrals. This service is optional in a name server, and the name server may also choose to restrict the clients which can use recursive mode.

Recursive service is helpful in several situations:

- a relatively simple requester that lacks the ability to use anything other than a direct answer to the question.

- a request that needs to cross protocol or other boundaries and can be sent to a server which can act as intermediary.

- a network where we want to concentrate the cache rather than having a separate cache for each client.

Non-recursive service is appropriate if the requester is capable of pursuing referrals and interested in information which will aid future requests.

The use of recursive mode is limited to cases where both the client and the name server agree to its use. The agreement is negotiated through the use of two bits in query and response messages:

- The recursion available, or RA bit, is set or cleared by a name server in all responses. The bit is true if the name server is willing to provide recursive service for the client, regardless of whether the client requested recursive service. That is, RA signals availability rather than use.

- Queries contain a bit called recursion desired or RD. This bit specifies specifies whether the requester wants recursive service for this query. Clients may request recursive service from any name server, though they should depend upon receiving it only from servers which have previously sent an RA, or servers which have agreed to provide service through private agreement or some other means outside of the DNS protocol.

The recursive mode occurs when a query with RD set arrives at a server which is willing to provide recursive service; the client can verify that recursive modewas used by checking that both RA and RD are set in the reply. Note that the name server should never perform recursive service unless asked via RD, since this interferes with trouble shooting of name servers and their databases.

If recursive service is requested and available, the recursive response to a query will be one of the following:

- The answer to the query, possibly preface by one or more CNAME RRs that specify aliases encountered on the way to an answer.
- A name error indicating that the name does not exist. This may include CNAME RRs that indicate that the original query name was an alias for a name which does not exist.
- A temporary error indication.

If recursive service is not requested or is not available, the non-recursive response will be one of the following:

- An authoritative name error indicating that the name does not exist.
- A temporary error indication.
- Some combination of:

RRs that answer the question, together with an indication whether the data comes from a zone or is cached.

A referral to name servers which have zones which are closer ancestors to the name than the server sending the reply.

- RRs that the name server thinks will prove useful to the requester.

Standard Query Processing

When processing queries with QCLASS=*, or some other QCLASS which matches multiple classes, the response should never be authoritative unless the server can guarantee that the response covers all classes.

When composing a response, RRs which are to be inserted in the additional section, but duplicate RRs in the answer or authority sections, may be omitted from the additional section.

When a response is so long that truncation is required, the truncation should start at the end of the response and work forward in the datagram. Thus if there is any data for the authority section, the answer section is guaranteed to be unique.

The MINIMUM value in the SOA should be used to set a floor on the TTL of data distributed from a zone. This floor function should be done when the data is copied into a response. This will allow future dynamic update protocols to change the SOA MINIMUM field without ambiguous semantics.

Zone Refresh and Reload Processing

In spite of a server's best efforts, it may be unable to load zone data from a master file due to syntax errors, etc., or be unable to refresh a zone within the its expiration parameter. In this case, the name server should answer queries as if it were not supposed to possess the zone.

If a master is sending a zone out via AXFR, and a new version is created during the transfer, the master should continue to send the old version if possible. In any case, it should never send part of one version and part of another. If completion is not possible, the master should reset the connection on which the zone transfer is taking place.

Inverse Queries (Optional)

Inverse queries are an optional part of the DNS. Name servers are not required to support any form of inverse queries. If a name server receives an inverse query that it does not support, it returns an error response with the "Not Implemented" error set in the header. While inverse query support is optional, all name servers must be at least able to return the error response.

RFC-1034/5 Domains: Name Server Inverse Queries

Contents of Inverse Queries and Responses

Inverse queries reverse the mappings performed by standard query operations; while a standard query maps a domain name to a resource, an inverse query maps a resource to a domain name. For example, a standard query might bind a domain name to a host address; the corresponding inverse query binds the host address to a domain name.

Inverse queries take the form of a single RR in the answer section of the message, with an empty question section. The owner name of the query RR and its TTL are not significant. The response carries questions in the question section which identify all names possessing the query RR WHICH THE NAME SERVER KNOWS. Since no name server knows about all of the domain name space, the response can never be assumed to be complete. Thus inverse queries are primarily useful for database management and debugging activities. Inverse queries are NOT an acceptable method of mapping host addresses to host names; use the IN- ADDR.ARPA domain instead.

Where possible, name servers should provide case-insensitive comparisons for inverse queries. Thus an inverse query asking for an MX RR of "Venera.isi.edu" should get the same response as a query for "VENERA.ISI.EDU"; an inverse query for HINFO RR "IBM-PC UNIX" should produce the same result as an inverse query for "IBM-pc unix". However, this cannot be guaranteed because name servers may possess RRs that contain character strings but the name server does not know that the data is character.

When a name server processes an inverse query, it either returns:

1. zero, one, or multiple domain names for the specified resource as QNAMEs in the question section

2. an error code indicating that the name server doesn't support inverse mapping of the specified resource type.

When the response to an inverse query contains one or more QNAMEs, the owner name and TTL of the RR in the answer section which defines the inverse query is modified to exactly match an RR found at the first QNAME.

RRs returned in the inverse queries cannot be cached using the same mechanism as is used for the replies to standard queries. One reason for this is that a name might have multiple RRs of the same type, and only one would appear. For example, an inverse query for a single address of a multiply homed host might create the impression that only one address existed.

RFC-1034/5 Domains: Name Server Inverse Queries

Query and Response Example

The overall structure of an inverse query for retrieving the domain name that corresponds to Internet address `10.1.0.52` is shown below:

Header	OPCODE=IQUERY, ID=997
Question	<empty>
Answer	<anyname> A IN 10.1.0.52
Authority	<empty>
Additional	<empty>

This query asks for a question whose answer is the Internet style address `10.1.0.52`. Since the owner name is not known, any domain name can be used as a placeholder (and is ignored). A single octet of zero, signifying the root, is usually used because it minimizes the length of the message. The TTL of the RR is not significant. The response to this query might be:

Header	OPCODE=RESPONSE, ID=997
Question	QTYPE=A,QCLASS=IN, QNAME=VENERA.ISI.EDU
Answer	VENERA.ISI.EDU A IN 10.10.52
Authority	<empty>
Additional	<empty>

Note that the QTYPE in a response to an inverse query is the same as the TYPE field in the answer section of the inverse query. Responses to inverse queries may contain multiple questions when the inverse is not unique. If the question section in the response is not empty, then the RR in the answer section is modified to correspond to be an exact copy of an RR at the first QNAME.

Processing

Name servers that support inverse queries can support these operations through exhaustive searches of their databases, but this becomes impractical as the size of the database increases. An alternative approach is to invert the database according to the search key.

For name servers that support multiple zones and a large amount of data, the recommended approach is separate inversions for each zone. When a particular zone is changed during a refresh, only its inversions need to be redone.

Support for transfer of this type of inversion may be included in future versions of the domain system, but is not supported in this version.

Algorithm

The actual algorithm used by the name server will depend on the local OS and data structures used to store RRs. The following algorithm assumes that the RRs are organized in several tree structures, one for each zone, and another for the cache:

(1) Set or clear the value of recursion available in the response depending on whether the name server is willing to provide recursive service. If recursive service is available and requested via the RD bit in the query, go to step 5, otherwise step 2.

(2) Search the available zones for the zone which is the nearest ancestor to QNAME. If such a zone is found, go to step 3, otherwise step 4.

(3) Start matching down, label by label, in the zone. The matching process can terminate several ways:

(a) If the whole of QNAME is matched, we have found the node.

If the data at the node is a CNAME, and QTYPE doesn't match CNAME, copy the CNAME RR into the answer section of the response, change QNAME to the canonical name in the CNAME RR, and go back to step 1.

Otherwise, copy all RRs which match QTYPE into the answer section and go to step 6.

(b) If a match would take us out of the authoritative data, we have a referral. This happens when we encounter a node with NS RRs marking cuts along the bottom of a zone.

Copy the NS RRs for the subzone into the authority section of the reply. Put whatever addresses are available into the additional section, using glue RRs if the addresses are not available from authoritative data or the cache. Go to step 4.

(c) If at some label, a match is impossible (i.e., the corresponding label does not exist), look to see if a the * label exists.

If the * label does not exist, check whether the name we are looking for is the original QNAME in the query or a name we have followed due to a CNAME. If the name is original, set an authoritative name error in the response and exit. Otherwise just exit.

If the * label does exist, match RRs at that node against QTYPE. If any match, copy them into the answer section, but set the owner of the RR to be QNAME, and not the node with the * label. Go to step 6.

(4) Start matching down in the cache. If QNAME is found in the cache, copy all RRs attached to it that match QTYPE into the answer section. If there was no delegation from authoritative data, look for the best one from the cache, and put it in the authority section. Go to step 6.

(5) Using the local resolver or a copy of its algorithm (see resolver section of this memo) to answer the query. Store the results, including any intermediate CNAMEs, in the answer section of the response.

(6) Using local data only, attempt to add other RRs which may be useful to the additional section of the query. Exit.

Wildcards

In the previous algorithm, special treatment was given to RRs with owner names starting with the label *. Such RRs are called wildcards. Wildcard RRs can be thought of as instructions for synthesizing RRs. When the appropriate conditions are met, the name server creates RRs with an owner name equal to the query name and contents taken from the wildcard RRs.

This facility is most often used to create a zone which will be used to forward mail from the Internet to some other mail system. The general idea is that anyname in that zone which is presented to server in a query will be assumed to exist, with certain properties, unless explicit evidence exists to the contrary. Note that the use of the term zone here, instead of domain, is intentional; such defaults do not propagate across zone boundaries, although a subzone may choose to achieve that appearance by setting up similar defaults.

The contents of the wildcard RRs follows the usual rules and formats for RRs. The wildcards in the zone have an owner name that controls the query names they will match. The owner name of the wildcard RRs is of the form "`*.<anydomain>`", where `<anydomain>` is any domain name. `<anydomain>` should not contain other * labels, and should be in the authoritative data of the zone. The wildcards potentially apply to descendants of `<anydomain>`, but not to `<anydomain>` itself. Another way to look at this is that the * label always matches at least one whole label and sometimes more, but always whole labels.

Wildcard RRs do not apply:

- When the query is in another zone. That is, delegation cancels the wildcard defaults.

- When the query name or a name between the wildcard domain and the query name is know to exist. For example, if a wildcard RR has an owner name of "`*.X`", and the zone also contains RRs attached to `B.X`, the wildcards would apply to queries for name `Z.X` (presuming there is no explicit information for `Z.X`), but not to `B.X`, `A.B.X`, or `X`.

A * label appearing in a query name has no special effect, but can be used to test for wildcards in an authoritative zone; such a query is the only way to get a response containing RRs with an owner name with * in it. The result of such a query should not be cached.

Note that the contents of the wildcard RRs are not modified when used to synthesize RRs.

To illustrate the use of wildcard RRs, suppose a large company with a large, non-IP/TCP, network wanted to create a mail gateway. If the company was called X.COM, and IP/TCP capable gateway machine was called A.X.COM, the following RRs might be entered into the COM zone:

```
X.COM           MX      10      A.X.COM

*.X.COM         MX      10      A.X.COM

A.X.COM         A       1.2.3.4
A.X.COM         MX      10      A.X.COM

*.A.X.COM       MX      10      A.X.COM
```

This would cause any MX query for any domain name ending in X.COM to return an MX RR pointing at A.X.COM. Two wildcard RRs are required since the effect of the wildcard at *.X.COM is inhibited in the A.X.COM subtree by the explicit data for A.X.COM. Note also that the explicit MX data at X.COM and A.X.COM is required, and that none of the RRs above would match a query name of XX.COM.

Negative Response Caching (Optional)

The DNS provides an optional service which allows name servers to distribute, and resolvers to cache, negative results with TTLs. For example, a name server can distribute a TTL along with a name error indication, and a resolver receiving such information is allowed to assume that the name does not exist during the TTL period without consulting authoritative data. Similarly, a resolver can make a query with a QTYPE which matches multiple types, and cache the fact that some of the types are not present.

This feature can be particularly important in a system which implements naming shorthands that use search lists beacuse a popular shorthand, which happens to require a suffix toward the end of the search list, will generate multiple name errors whenever it is used.

The method is that a name server may add an SOA RR to the additional section of a response when that response is authoritative. The SOA must be that of the zone which was the source of the authoritative data in the answer section, or name error if applicable. The MINIMUM field of the SOA controls the length of timethat the negative result may be cached.

Note that in some circumstances, the answer section may contain multiple owner names. In this case, the SOA mechanism should only be used for the data which matches QNAME, which is the only authoritative data in this section.

Name servers and resolvers should never attempt to add SOAs to the additional section of a non-authoritative response, or attempt to infer results which are not directly stated in an authoritative response. There are several reasons for this, including: cached information isn't usually enough to match up RRs and their zone names, SOA RRs may be cached due to direct SOA queries, and name servers are not required to output the SOAs in the authority section.

This feature is optional, although a refined version is expected to become part of the standard protocol in the future. Name servers are not required to add the SOA RRs in all authoritative responses, nor are resolvers required to cache negative results. Both are recommended. All resolvers and recursive name servers are required to at least be able to ignore the SOA RR when it is present in a response.

Some experiments have also been proposed which will use this feature. The idea is that if cached data is known to come from a particular zone, and if an authoritative copy of the zone's SOA is obtained, and if the zone's SERIAL has not changed since the data was cached, then the TTL of the cached data can be reset to the zone MINIMUM value if it is smaller. This usage is mentioned for planning purposes only, and is not recommended as yet.

RFC-1034/5 Domain System: Name Servers

Zone Maintenance and Transfers

Part of the job of a zone administrator is to maintain the zones at all of the name servers which are authoritative for the zone. When the inevitable changes are made, they must be distributed to all of the name servers. While this distribution can be accomplished using FTP or some other ad hoc procedure, the preferred method is the zone transfer part of the DNS protocol.

The general model of automatic zone transfer or refreshing is that one of the name servers is the master or primary for the zone. Changes are coordinated at the primary, typically by editing a master file for the zone. After editing, the administrator signals the master server to load the new zone. The other non-master or secondary servers for the zone periodically check for changes (at a selectable interval) and obtain new zone copies when changes have been made.

To detect changes, secondaries just check the SERIAL field of the SOA for the zone. In addition to whatever other changes are made, the SERIAL field in the SOA of the zone is always advanced whenever any change is made to the zone. The advancing can be a simple increment, or could be based on the write date and time of the master file, etc. The purpose is to make it possible to determine which of two copies of a zone is more recent by comparing serial numbers. Serial number advances and comparisons use sequence space arithmetic, so there is a theoretic limit on how fast a zone can be updated, basically that old copies must die out before the serial number covers half of its 32 bit range. In practice, the only concern is that the compare operation deals properly with comparisons around the boundary between the most positive and most negative 32 bit numbers.

The periodic polling of the secondary servers is controlled by parameters in the SOA RR for the zone, which set the minimum acceptable polling intervals. The parameters are called REFRESH, RETRY, and EXPIRE. Whenever a new zone is loaded in a secondary, the secondary waits REFRESH seconds before checking with the primary for a new serial. If this check cannot be completed, new checks are started every RETRY seconds. The check is a simple query to the primary for the SOA RR of the zone. If the serial field in the secondary's zone copy is equal to the serial returned by the primary, then no changes have occurred, and the REFRESH interval wait is restarted. If the secondary finds it impossible to perform a serial check for the EXPIRE interval, it must assume that its copy of the zone is obsolete an discard it.

When the poll shows that the zone has changed, then the secondary server must request a zone transfer via an AXFR request for the zone. The AXFR may cause an error, such as refused, but normally is answered by a sequence of response messages. The first and last messages must contain the data for the top authoritative node of the zone. Intermediate messages carry all of the other RRs from the zone, including both authoritative and non-authoritative RRs. The stream of messages allows the secondary to construct a copy of the zone. Because accuracy is essential, TCP or some other reliable protocol must be used for AXFR requests.

Each secondary server is required to perform the following operations against the master, but may also optionally perform these operations against other secondary servers. This strategy can improve the transfer process when the primary is unavailable due to host downtime or network problems, or when a secondary server has better network access to an "intermediate" secondary than to the primary.

Resolvers

RFC-1034/5 Domain Name System: Resolvers

Introduction

Resolvers are programs that interface user programs to domain name servers. In the simplest case, a resolver receives a request from a user program (e.g., mail programs, TELNET, FTP) in the form of a subroutine call, system call etc., and returns the desired information in a form compatible with the local host's data formats.

The resolver is located on the same machine as the program that requests the resolver's services, but it may need to consult name servers on other hosts. Because a resolver may need to consult several name servers, or may have the requested information in a local cache, the amount of time that a resolver will take to complete can vary quite a bit, from milliseconds to several seconds.

A very important goal of the resolver is to eliminate network delay and name server load from most requests by answering them from its cache of prior results. It follows that caches which are shared by multiple processes, users, machines, etc., are more efficient than non-shared caches.

Typical Functions

The client interface to the resolver is influenced by the local host's conventions, but the typical resolver-client interface has three functions:

(1) Host name to host address translation.

This function is often defined to mimic a previous `HOSTS.TXT` based function. Given a character string, the caller wants one or more 32 bit IP addresses. Under the DNS, it translates into a request for type A RRs. Since the DNS does not preserve the order of RRs, this function may choose to sort the returned addresses or select the "best" address if the service returns only one choice to the client. Note that a multiple address return is recommended, but a single address may be the only way to emulate prior `HOSTS.TXT` services.

(2) Host address to host name translation

This function will often follow the form of previous functions. Given a 32 bit IP address, the caller wants a character string. The octets of the IP address are reversed, used as name components, and suffixed with "`IN-ADDR.ARPA`". A type PTR query is used to get the RR with the primary name of the host. For example, a request for the host name corresponding to IP address `1.2.3.4` looks for PTR RRs for domain name "`4.3.2.1.IN-ADDR.ARPA`".

(3) General lookup function

This function retrieves arbitrary information from the DNS, and has no counterpart in previous systems. The caller supplies a QNAME, QTYPE, and QCLASS, and wants all of the matching RRs. This function will often use the DNS format for all RR data instead of the local host's, and returns all RR content (e.g., TTL) instead of a processed form with local quoting conventions.

When the resolver performs the indicated function, it usually has one of the following results to pass back to the client:

- One or more RRs giving the requested data.

  In this case the resolver returns the answer in the appropriate format.

- A name error (NE).

  This happens when the referenced name does not exist. For example, a user may have mistyped a host name.

- A data not found error.

  This happens when the referenced name exists, but data of the appropriate type does not. For example, a host address function applied to a mailbox name would return this error since the name exists, but no address RR is present.

It is important to note that the functions for translating between host names and addresses may combine the "name error" and "data not found" error conditions into a single type of error return, but the general function should not. One reason for this is that applications may ask first for one type of information about a name followed by a second request to the same name for some other type of information; if the two errors are combined, then useless queries may slow the application.

Aliases

While attempting to resolve a particular request, the resolver may find that the name in question is an alias. For example, the resolver might find that the name given for host name to address translation is an alias when it finds the CNAME RR. If possible, the alias condition should be signalled back from the resolver to the client.

In most cases a resolver simply restarts the query at the new name when it encounters a CNAME. However, when performing the general function, the resolver should not pursue aliases when the CNAME RR matches the query type. This allows queries which ask whether an alias is present. For example, if the query type isCNAME, the user is interested in the CNAME RR itself, and not the RRs at the name it points to.

Several special conditions can occur with aliases. Multiple levels of aliases should be avoided due to their lack of efficiency, but should not be signalled as an error. Alias loops and aliases which point to non-existent names should be caught and an error condition passed back to the client.

Temporary Failures

In a less than perfect world, all resolvers will occasionally be unable to resolve a particular request. This condition can be caused by a resolver which becomes separated from the rest of the network due to a link failure or gateway problem, or less often by coincident failure or unavailability of all servers for aparticular domain.

It is essential that this sort of condition should not be signalled as a name or data not present error to applications. This sort of behavior is annoying to humans, and can wreak havoc when mail systems use the DNS.

While in some cases it is possible to deal with such a temporary problem by blocking the request indefinitely, this is usually not a good choice, particularly when the client is a server process that could move on to other tasks. The recommended solution is to always have temporary failure as one of the possible results of a resolver function, even though this may make emulation of existing `HOSTS.TXT` functions more difficult.

Resolver Internals

Every resolver implementation uses slightly different algorithms, and typically spends much more logic dealing with errors of various sorts than typical occurances. This section outlines a recommended basic strategy for resolver operation.

Stub Resolvers

One option for implementing a resolver is to move the resolution function out of the local machine and into a name server which supports recursive queries. This can provide an easy method of providing domain service in a PC which lacks the resources to perform the resolver function, or can centralize the cache for awhole local network or organization.

All that the remaining stub needs is a list of name server addresses that will perform the recursive requests. This type of resolver presumably needs the information in a configuration file, since it probably lacks the sophistication to locate it in the domain database. The user also needs to verify that the listedservers will perform the recursive service; a name server is free to refuse to perform recursive services for any or all clients. The user should consult the local system administrator to find name servers willing to perform the service.

This type of service suffers from some drawbacks. Since the recursive requests may take an arbitrary amount of time to perform, the stub may have difficulty optimizing retransmission intervals to deal with both lost UDP packets and dead servers; the name server can be easily overloaded by too zealous a stub if it interprets retransmissions as new requests. Use of TCP may be an answer, but TCP may well place burdens on the host's capabilities which are similar to those ofa real resolver.

Resources

In addition to its own resources, the resolver may also have shared access to zones maintained by a local name server. This gives the resolver the advantage of more rapid access, but the resolver must be careful to never let cached information override zone data. In this discussion the term "local information" is meant to mean the union of the cache and such shared zones, with the understanding that authoritative data is always used in preference to cached data when bothare present.

The following resolver algorithm assumes that all functions have been converted to a general lookup function, and uses the following data structures to represent the state of a request in progress in the resolver:

SNAME the domain name we are searching for.

STYPE the QTYPE of the search request.

SCLASS the QCLASS of the search request.

SLIST a structure which describes the name servers and the zone which the resolver is currently trying to query. This structure keeps track of the resolver's current best guess about which name servers hold the desired information; it is updated when arriving information changes the guess. This structure includes the equivalent of a zone name, the known name servers for the zone, the known addresses for the name servers, and history information which can be used to suggest which server is likely to be the best one to try next. The zone name equivalent is a match count of the number of labels from the root down which SNAME has in common with the zone being queried; this is used as a measure of how "close" the resolver is to SNAME.

SBELT a "safety belt" structure of the same form as SLIST, which is initialized from a configuration file, and lists servers which should be used when the resolver doesn't have any local information to guide name server selection. The match count will be -1 to indicate that no labels are known to match.

CACHE A structure which stores the results from previous responses. Since resolvers are responsible for discarding old RRs whose TTL has expired, most implementations convert the interval specified in arriving RRs to some sort of absolute time when the RR is stored in the cache. Instead of counting the TTLs down individually, the resolver just ignores or discards old RRs when it runs across them in the course of a search, or discards them during periodic sweeps to reclaim the memory consumed by old RRs.

Algorithm

The top level algorithm has four steps:

(1) See if the answer is in local information, and if so return it to the client.

(2) Find the best servers to ask.

(3) Send them queries until one returns a response.

(4) Analyze the response, either:

(a) if the response answers the question or contains a name error, cache the data as well as returning it back to the client.

(b) if the response contains a better delegation to other servers, cache the delegation information, and go to step 2.

(c) if the response shows a CNAME and that is not the answer itself, cache the CNAME, change the SNAME to the canonical name in the CNAME RR and go to step 1.

(d) if the response shows a servers failure or other bizarre contents, delete the server from the SLIST and go back to step 3.

Step 1 searches the cache for the desired data. If the data is in the cache, it is assumed to be good enough for normal use. Some resolvers have an option at the user interface which will force the resolver to ignore the cached data and consult with an authoritative server. This is not recommended as the default. If the resolver has direct access to a name server's zones, it should check to see if the desired data is present in authoritative form, and if so, use the authoritative data in preference to cached data.

Step 2 looks for a name server to ask for the required data. The general strategy is to look for locally-available name server RRs, starting at SNAME, then the parent domain name of SNAME, the grandparent, and so on toward the root. Thus if SNAME were `Mockapetris.ISI.EDU`, this step would look for NS RRs for `Mockapetris.ISI.EDU`, then `ISI.EDU`, then `EDU`, and then . (the root). These NS RRs list the names of hosts for a zone at or above SNAME. Copy the names into SLIST. Set up their addresses using local data. It may be the case that the addresses are not available. The resolver has many choices here; the best is to start parallel resolver processes looking for the addresses while continuing onward with the addresses which are available. Obviously, the design choices and options are complicated and a function of the local host's capabilities. The recommended priorities for the resolver designer are:

(1) Bound the amount of work (packets sent, parallel processes started) so that a request can't get into an infinite loop or start off a chain reaction of requests or queries with other implementations EVEN IF SOMEONE HAS INCORRECTLY CONFIGURED SOME DATA.

(2) Get back an answer if at all possible.

(3) Avoid unnecessary transmissions.

(4) Get the answer as quickly as possible.

If the search for NS RRs fails, then the resolver initializes SLIST from the safety belt SBELT. The basic idea is that when the resolver has no idea what servers to ask, it should use information from a configuration file that lists several servers which are expected to be helpful. Although there are special situations, the usual choice is two of the root servers and two of the servers for the host's domain. The reason for two of each is for redundancy. The root servers will provide eventual access to all of the domain space. The two local servers will allow the resolver to continue to resolve local names if the local network becomes isolated from the internet due to gateway or link failure.

In addition to the names and addresses of the servers, the SLIST data structure can be sorted to use the best servers first, and to insure that all addresses of all servers are used in a round-robin manner. The sorting can be a simple function of preferring addresses on the local network over others, or may involve statistics from past events, such as previous response times and batting averages.

Step 3 sends out queries until a response is received. The strategy is to cycle around all of the addresses for all of the servers with a timeout between eachtransmission. In practice it is important to use all addresses of a multihomed host, and too aggressive a retransmission policy actually slows response when used by multiple resolvers contending for the same name server and even occasionally for a single resolver. SLIST typically contains data values to control thetimeouts and keep track of previous transmissions.

Step 4 involves analyzing responses. The resolver should be highly paranoid in its parsing of responses. It should also check that the response matches the query it sent using the ID field in the response.

The ideal answer is one from a server authoritative for the query which either gives the required data or a name error. The data is passed back to the user and entered in the cache for future use if its TTL is greater than zero.

If the response shows a delegation, the resolver should check to see that the delegation is "closer" to the answer than the servers in SLIST are. This can be done by comparing the match count in SLIST with that computed from SNAME and the NS RRs in the delegation. If not, the reply is bogus and should be ignored. If the delegation is valid the NS delegation RRs and any address RRs for the servers should be cached. The name servers are entered in the SLIST, and the search is restarted.

If the response contains a CNAME, the search is restarted at the CNAME unless the response has the data for the canonical name or if the CNAME is the answer itself.

Resolver Implementation

This section discusses implementation details assuming the database structure suggested in the name server implementation section of this memo.

Transforming a User Request Into a Query

The first step a resolver takes is to transform the client's request, stated in a format suitable to the local OS, into a search specification for RRs at a specific name which match a specific QTYPE and QCLASS. Where possible, the QTYPE and QCLASS should correspond to a single type and a single class, because this makes the use of cached data much simpler. The reason for this is that the presence of data of one type in a cache doesn't confirm the existence or non-existence of data of other types, hence the only way to be sure is to consult an authoritative source. If QCLASS=* is used, then authoritative answers won't be available.

Since a resolver must be able to multiplex multiple requests if it is to perform its function efficiently, each pending request is usually represented in some block of state information. This state block will typically contain:

- A timestamp indicating the time the request began. The timestamp is used to decide whether RRs in the database can be used or are out of date. This timestamp uses the absolute time format previously discussed for RR storage in zones and caches. Note that when an RRs TTL indicates a relative time, the RR must be timely, since it is part of a zone. When the RR has an absolute time, it is part of a cache, and the TTL of the RR is compared against the timestamp for the start of the request.

- Note that using the timestamp is superior to using a current time, since it allows RRs with TTLs of zero to be entered in the cache in the usual manner, but still used by the current request, even after intervals of many seconds due to system load, query retransmission timeouts, etc.

- Some sort of parameters to limit the amount of work which will be performed for this request.

- The amount of work which a resolver will do in response to a client request must be limited to guard against errors in the database, such as circular CNAME references, and operational problems, such as network partition which prevents the resolver from accessing the name servers it needs. While local limits on the number of times a resolver will retransmit a particular query to a particular name server address are essential, the resolver should have a global per-request counter to limit work on a single request. The counter should be set to some initial value and decremented whenever the resolver performs any action (retransmission timeout, retransmission, etc.) If the counter passes zero, the request is terminated with a temporary error.

- Note that if the resolver structure allows one request to start others in parallel, such as when the need to access a name server for one request causes a parallel resolve for the name server's addresses, the spawned request should be started with a lower counter. This prevents circular references in the database from starting a chain reaction of resolver activity.

Sending Queries

The basic task of the resolver is to formulate a query which will answer the client's request and direct that query to name servers which can provide the information. The resolver will usually only have very strong hints about which servers to ask, in the form of NS RRs, and may have to revise the query, in response to CNAMEs, or revise the set of name servers the resolver is asking, in response to delegation responses which point the resolver to name servers closer to the desired information. In addition to the information requested by the client, the resolver may have to call upon its own services to determine the address of name servers it wishes to contact.

In any case, the model used in this memo assumes that the resolver is multiplexing attention between multiple requests, some from the client, and some internally generated. Each request is represented by some state information, and the desired behavior is that the resolver transmit queries to name servers in a way that maximizes the probability that the request is answered, minimizes the time that the request takes, and avoids excessive transmissions. The key algorithm uses the state information of the request to select the next name server address to query, and also computes a timeout which will cause the next action should a response not arrive. The next action will usually be a transmission to some other server, but may be a temporary error to the client.

The resolver always starts with a list of server names to query (SLIST). This list will be all NS RRs which correspond to the nearest ancestor zone that the resolver knows about. To avoid startup problems, the resolver should have a set of default servers which it will ask should it have no current NS RRs which are appropriate. The resolver then adds to SLIST all of the known addresses for the name servers, and may start parallel requests to acquire the addresses of the servers when the resolver has the name, but no addresses, for the name servers.

To complete initialization of SLIST, the resolver attaches whatever history information it has to the each address in SLIST. This will usually consist of some sort of weighted averages for the response time of the address, and the batting average of the address (i.e., how often the address responded at all to the request). Note that this information should be kept on a per address basis, rather than on a per name server basis, because the response time and batting average of a particular server may vary considerably from address to address. Note also that this information is actually specific to a resolver address / server address pair, so a resolver with multiple addresses may wish to keep separate histories for each of its addresses. Part of this step must deal with addresses which have no such history; in this case an expected round trip time of 5-10 seconds should be the worst case, with lower estimates for the same local network, etc.

Note that whenever a delegation is followed, the resolver algorithm reinitializes SLIST.

The information establishes a partial ranking of the available name server addresses. Each time an address is chosen and the state should be altered to prevent its selection again until all other addresses have been tried. The timeout for each transmission should be 50-100% greater than the average predicted value to allow for variance in response.

Some fine points:

- The resolver may encounter a situation where no addresses are available for any of the name servers named in SLIST, and where the servers in the list are precisely those which would normally be used to

look up their own addresses. This situation typically occurs when the glue address RRs have a smaller TTL than the NS RRs marking delegation, or when the resolver caches the result of a NS search. The resolver should detect this condition and restart the search at the next ancestor zone, or alternatively at the root.

- If a resolver gets a server error or other bizarre response from a name server, it should remove it from SLIST, and may wish to schedule an immediate transmission to the next candidate server address.

Processing Responses

The first step in processing arriving response datagrams is to parse the response. This procedure should include:

- Check the header for reasonableness. Discard datagrams which are queries when responses are expected.
- Parse the sections of the message, and insure that all RRs are correctly formatted.
- As an optional step, check the TTLs of arriving data looking for RRs with excessively long TTLs. If a RR has an excessively long TTL, say greater than 1 week, either discard the whole response, or limit all TTLs in the response to 1 week.

The next step is to match the response to a current resolver request. The recommended strategy is to do a preliminary matching using the ID field in the domain header, and then to verify that the question section corresponds to the information currently desired. This requires that the transmission algorithm devote several bits of the domain ID field to a request identifier of some sort. This step has several fine points:

- Some name servers send their responses from different addresses than the one used to receive the query. That is, a resolver cannot rely that a response will come from the same address which it sent the corresponding query to. This name server bug is typically encountered in UNIX systems.
- If the resolver retransmits a particular request to a name server it should be able to use a response from any of the transmissions. However, if it is using the response to sample the round trip time to access the name server, it must be able to determine which transmission matches the response (and keep transmission times for each outgoing message), or only calculate round trip times based on initial transmissions.
- A name server will occasionally not have a current copy of a zone which it should have according to some NS RRs. The resolver should simply remove the name server from the current SLIST, and continue.

Using the Cache

In general, we expect a resolver to cache all data which it receives in responses since it may be useful in answering future client requests. However, there are several types of data which should not be cached:

- When several RRs of the same type are available for a particular owner name, the resolver should either cache them all or none at all. When a response is truncated, and a resolver doesn't know whether it has a complete set, it should not cache a possibly partial set of RRs.

- Cached data should never be used in preference to authoritative data, so if caching would cause this to happen the data should not be cached.

- The results of an inverse query should not be cached.

- The results of standard queries where the QNAME contains "*" labels if the data might be used to construct wildcards. The reason is that the cache does not necessarily contain existing RRs or zone boundary information which is necessary to restrict the application of the wildcard RRs.

- RR data in responses of dubious reliability. When a resolver receives unsolicited responses or RR data other than that requested, it should discard it without caching it. The basic implication is that all sanity checks on a packet should be performed before any of it is cached.

In a similar vein, when a resolver has a set of RRs for some name in a response, and wants to cache the RRs, it should check its cache for already existing RRs. Depending on the circumstances, either the data in the response or the cache is preferred, but the two should never be combined. If the data in the response is from authoritative data in the answer section, it is always preferred.

Mail Support

RFC-1034/5 Domain System: Mail Support

Introduction

The domain system defines a standard for mapping mailboxes into domain names, and two methods for using the mailbox information to derive mail routing information. The first method is called mail exchange binding and the other method is mailbox binding. The mailbox encoding standard and mail exchange binding are part of the DNS official protocol, and are the recommended method for mail routing in the Internet. Mailbox binding is an experimental feature which is still under development and subject to change.

The mailbox encoding standard assumes a mailbox name of the form "<local-part>@<mail-domain>". While the syntax allowed in each of these sections varies substantially between the various mail internets, the preferred syntax for the ARPA Internet is given in (RFC-822).

The DNS encodes the <local-part> as a single label, and encodes the <mail-domain> as a domain name. The single label from the <local-part> is prefaced to the domain name from <mail-domain> to form the domain name corresponding to the mailbox. Thus the mailbox HOSTMASTER@SRI- NIC.ARPA is mapped into the domain name HOSTMASTER.SRI-NIC.ARPA. If the <local-part> contains dots or other special characters, its representation in a master file will require the use of backslash quoting to ensure that the domain name is properly encoded. For example, the mailbox Action.domains@ISI.EDU would be represented as Action\.domains.ISI.EDU.

Mail exchange binding

Mail exchange binding uses the <mail-domain> part of a mailbox specification to determine where mail should be sent. The <local-part> is not even consulted. RFC-974 on mail routing specifies this method in detail, and should be consulted before attempting to use mail exchange support.

One of the advantages of this method is that it decouples mail destination naming from the hosts used to support mail service, at the cost of another layer of indirection in the lookup function. However, the addition layer should eliminate the need for complicated "%", "!", etc encodings in <local-part>.

> The essence of the method is that the <mail-domain> is used as a domain name to locate type MX RRs which list hosts willing to accept mail for <mail-domain>, together with preference values which rank the hosts according to an order specified by the administrators for <mail-domain>.

In this memo, the <mail-domain> ISI.EDU is used in examples, together with the hosts VENERA.ISI.EDU and VAXA.ISI.EDU as mail exchanges for ISI.EDU. If a mailer had a message for Mockapetris@ISI.EDU, it would route it by looking up MX RRs for ISI.EDU. The MX RRs at ISI.EDU name VENERA.ISI.EDU and VAXA.ISI.EDU, and type A queries can find the host addresses.

RFC-1034/5 Domain System: Mail Support

Mailbox Binding (Experimental)

In mailbox binding, the mailer uses the entire mail destination specification to construct a domain name. The encoded domain name for the mailbox is used as the QNAME field in a `QTYPE=MAILB` query.

Several outcomes are possible for this query:

1. The query can return a name error indicating that the mailbox does not exist as a domain name.

   In the long term, this would indicate that the specified mailbox doesn't exist. However, until the use of mailbox binding is universal, this error condition should be interpreted to mean that the organization identified by the global part does not support mailbox binding. The appropriate procedure is to revert to exchange binding at this point.

2. The query can return a Mail Rename (MR) RR.

   The MR RR carries new mailbox specification in its RDATA field. The mailer should replace the old mailbox with the new one and retry the operation.

3. The query can return a MB RR.

   The MB RR carries a domain name for a host in its RDATA field. The mailer should deliver the message to that host via whatever protocol is applicable, e.g., b,SMTP.

4. The query can return one or more Mail Group (MG) RRs.

   This condition means that the mailbox was actually a mailing list or mail group, rather than a single mailbox. Each MG RR has a RDATA field that identifies a mailbox that is a member of the group. The mailer should deliver a copy of the message to each member.

5. The query can return a MB RR as well as one or more MG RRs.

   This condition means the the mailbox was actually a mailing list. The mailer can either deliver the message to the host specified by the MB RR, which will in turn do the delivery to all members, or the mailer can use the MG RRs to do the expansion itself.

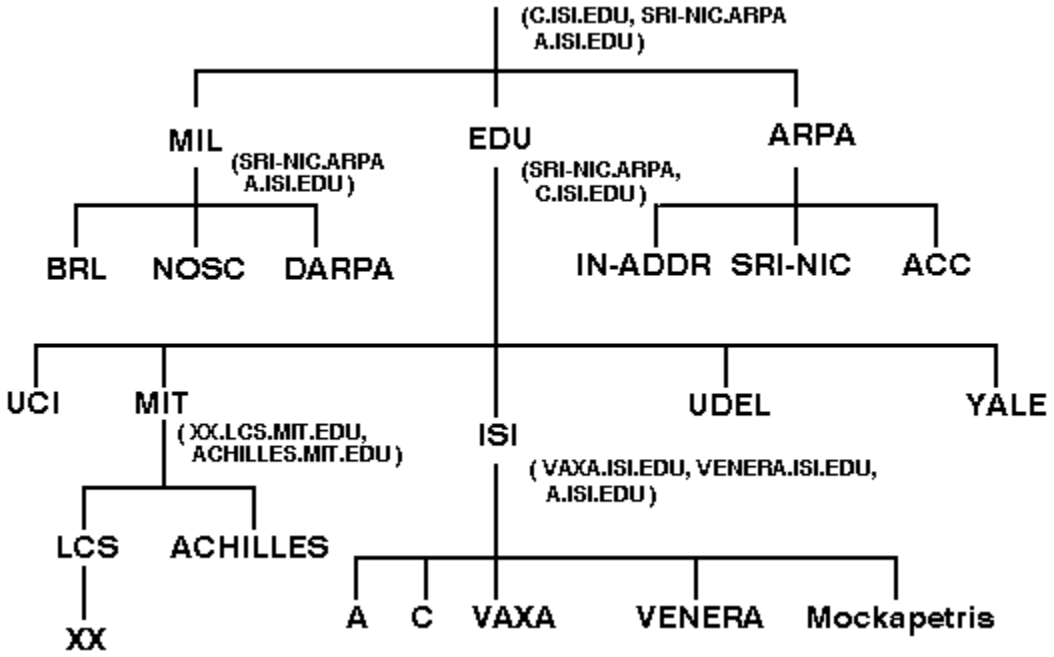
In any of these cases, the response may include a Mail Information (MINFO) RR. This RR is usually associated with a mail group, but is legal with a MB. The MINFO RR identifies two mailboxes. One of these identifies a responsible person for the original mailbox name. This mailbox should be used for requests to be added to a mail group, etc. The second mailbox name in the MINFO RR identifies a mailbox that should receive error messages for mail failures. This is particularly appropriate for mailing lists when errors in member names should be reported to a person other than the one who sends a message to the list.

New fields may be added to this RR in the future.

RFC-1034/5 Domain Name System

A Scenario

In our sample domain space, suppose we wanted separate administrative control for the root, `MIL`, `EDU`, `MIT.EDU` and `ISI.EDU` zones. We might allocate name servers as follows:



In this example, the authoritative name server is shown in parentheses at the point in the domain tree at which is assumes control.

Thus the root name servers are on `C.ISI.EDU`, `SRI-NIC.ARPA`, and `A.ISI.EDU`. The `MIL` domain is served by `SRI-NIC.ARPA` and `A.ISI.EDU`. The `EDU` domain is served by `SRI-NIC.ARPA`. and `C.ISI.EDU`. Note that servers may have zones which are contiguous or disjoint. In this scenario, `C.ISI.EDU` has contiguous zones at the root and `EDU` domains. `A.ISI.EDU` has contiguous zones at the root and MIL domains, but also has a non-contiguous zone at `ISI.EDU`.

C.ISI.EDU Name Server
Example Standard Queries
QNAME=SRI-NIC.ARPA, QTYPE=A
QNAME=SRI-NIC.ARPA, QTYPE=*
QNAME=SRI-NIC.ARPA, QTYPE=MX
QNAME=SRI-NIC.ARPA, QTYPE=NS
QNAME=SIR-NIC.ARPA, QTYPE=A
QNAME=BRL.MIL, QTYPE=A
QNAME=USC-ISIC.ARPA, QTYPE=A
QNAME=USC-ISIC.ARPA, QTYPE=CNAME
Example Resolution
Resolve MX for ISI.EDU.
Get the Host Name for Address 26.6.0.65
Get the Host Address of poneria.ISI.EDU

RFC-1034/5 Domain Name System: Scenario

C.ISI.EDU Name Server

C.ISI.EDU is a name server for the root, MIL, and EDU domains of the IN class, and would have zones for these domains. The zone data for the root domain might be:

```
.          IN      SOA     SRI-NIC.ARPA. HOSTMASTER.SRI-NIC.ARPA. (
                           870611          ;serial
                           1800            ;refresh every 30 min
                           300             ;retry every 5 min
                           604800          ;expire after a week
                           86400)          ;minimum of a day
                   NS      A.ISI.EDU.
                   NS      C.ISI.EDU.
                   NS      SRI-NIC.ARPA.

MIL.       86400   NS      SRI-NIC.ARPA.
           86400   NS      A.ISI.EDU.

EDU.       86400   NS      SRI-NIC.ARPA.
           86400   NS      C.ISI.EDU.

SRI-NIC.ARPA.      A       26.0.0.73
                   A       10.0.0.51
                   MX      0 SRI-NIC.ARPA.
                   HINFO   DEC-2060 TOPS20

ACC.ARPA.          A       26.6.0.65
                   HINFO   PDP-11/70 UNIX
                   MX      10 ACC.ARPA.

USC-ISIC.ARPA.     CNAME   C.ISI.EDU.

73.0.0.26.IN-ADDR.ARPA.    PTR    SRI-NIC.ARPA.
65.0.6.26.IN-ADDR.ARPA.    PTR    ACC.ARPA.
51.0.0.10.IN-ADDR.ARPA.    PTR    SRI-NIC.ARPA.
52.0.0.10.IN-ADDR.ARPA.    PTR    C.ISI.EDU.

103.0.3.26.IN-ADDR.ARPA. PTR    A.ISI.EDU.

A.ISI.EDU. 86400 A      26.3.0.103
C.ISI.EDU. 86400 A      10.0.0.52
```

This data is represented as it would be in a master file. Most RRs are single line entries; the sole exception here is the SOA RR, which uses "(" to start a multi-line RR and ")" to show the end of a multi-line RR. Since the class of all RRs in a zone must be the same, only the first RR in a zone need specify the class. When a name server loads a zone, it forces the TTL of all authoritative RRs to be at least the MINIMUM field of the SOA, here 86400 seconds, or one day.The NS RRs marking delegation of the MIL and EDU domains, together with the glue RRs for the servers host addresses, are not part of the authoritative data in the zone, and hence have explicit TTLs.

Four RRs are attached to the root node: the SOA which describes the root zone and the 3 NS RRs which list the name servers for the root. The data in the SOA RR describes the management of the zone. The zone data is maintained on host SRI-NIC.ARPA, and the

responsible party for the zone is HOSTMASTER@SRI-NIC.ARPA. A key item in the SOA is the 86400 second minimum TTL, which means that all authoritative data in the zone has at least that TTL, although higher values may beexplicitly specified.

The NS RRs for the MIL and EDU domains mark the boundary between the root zone and the MIL and EDU zones. Note that in this example, the lower zones happen tobe supported by name servers which also support the root zone.

The master file for the EDU zone might be stated relative to the origin EDU. The zone data for the EDU domain might be:

```
EDU.  IN SOA SRI-NIC.ARPA. HOSTMASTER.SRI-NIC.ARPA. (
                        870729 ;serial
                        1800 ;refresh every 30 minutes
                        300 ;retry every 5 minutes
                        604800 ;expire after a week
                        86400 ;minimum of a day
                        )
                NS SRI-NIC.ARPA.
                NS C.ISI.EDU.

UCI 172800 NS ICS.UCI
                172800 NS ROME.UCI
ICS.UCI 172800 A 192.5.19.1
ROME.UCI 172800 A 192.5.19.31

ISI 172800 NS VAXA.ISI
                172800 NS A.ISI
                172800 NS VENERA.ISI.EDU.
VAXA.ISI 172800 A 10.2.0.27
                172800 A 128.9.0.33
VENERA.ISI.EDU. 172800 A 10.1.0.52
                172800 A 128.9.0.32
A.ISI 172800 A 26.3.0.103

UDEL.EDU.  172800 NS LOUIE.UDEL.EDU.
                172800 NS UMN-REI-UC.ARPA.
LOUIE.UDEL.EDU. 172800 A 10.0.0.96
                172800 A 192.5.39.3

YALE.EDU.  172800 NS YALE.ARPA.
YALE.EDU.  172800 NS YALE-BULLDOG.ARPA.

MIT.EDU.  43200 NS XX.LCS.MIT.EDU.
                  43200 NS ACHILLES.MIT.EDU.
XX.LCS.MIT.EDU.  43200 A 10.0.0.44
ACHILLES.MIT.EDU. 43200 A 18.72.0.8
```

Note the use of relative names here. The owner name for the ISI.EDU. is stated using a relative name, as are two of the name server RR contents. Relative and absolute domain names may be freely intermixed in a master

Example Standard Queries

The following queries and responses illustrate name server behavior. Unless otherwise noted, the queries do not have recursion desired (RD) in the header. Note that the answers to non-recursive queries do depend on the server being asked, but do not depend on the identity of the requester.

RFC-1034/5 Domain Name System: Scenario

QNAME=SRI-NIC.ARPA, QTYPE=A

The query would look like:

Header	OPCODE=SQUERY
Question	QNAME=SRI-NIC.ARPA., QCLASS=IN, QTYPE=A
Answer	<empty>
Authority	<empty>
Additional	<empty>

The response from `C.ISI.EDU` would be:

Header	OPCODE=SQUERY, RESPONSE, AA
Question	QNAME=SRI-NIC.ARPA., QCLASS=IN, QTYPE=A
Answer	SRI-NIC.ARPA. 86400 IN A 26.0.0.73 86400 IN A 10.0.0.51
Authority	<empty>
Additional	<empty>

The header of the response looks like the header of the query, except that the RESPONSE bit is set, indicating that this message is a response, not a query, and the Authoritative Answer (AA) bit is set indicating that the address RRs in the answer section are from authoritative data. The question section of the response matches the question section of the query.

If the same query was sent to some other server which was not authoritative for `SRI-NIC.ARPA`, the response might be:

Header	OPCODE=SQUERY, RESPONSE
Question	QNAME=SRI-NIC.ARPA., QCLASS=IN, QTYPE=A
Answer	SRI-NIC.ARPA. 1777 IN A 10.0.0.51 1777 IN A 26.0.0.73
Authority	<empty>
Additional	<empty>

This response is different from the previous one in two ways: the header does not have AA set, and the TTLs are different. The inference is that the data did not come from a zone, but from a cache. The difference between the authoritative TTL and the TTL here is due to aging of the data in a cache. The difference in ordering of the RRs in the answer section is not significant.

RFC-1034/5 Domain Name System: Scenario

QNAME=SRI-NIC.ARPA, QTYPE=*

A query similar to the previous one, but using a QTYPE of *, would receive the following response from `C.ISI.EDU`:

Header	OPCODE=SQUERY, RESPONSE, AA
Question	QNAME=SRI-NIC.ARPA., QCLASS=IN, QTYPE=*
Answer	SRI-NIC.ARPA. 86400 IN A 26.0.0.73 A 10.0.0.51 MX 0 SRI-NIC.ARPA. HINFO DEC-2060 TOPS20
Authority	<empty>
Additional	<empty>

If a similar query was directed to two name servers which are not authoritative for `SRI-NIC.ARPA`, the responses might be:

Header	OPCODE=SQUERY, RESPONSE
Question	QNAME=SRI-NIC.ARPA., QCLASS=IN, QTYPE=*
Answer	SRI-NIC.ARPA. 12345 in A 26.0.0.73 A 10.0.0.51
Authority	<empty>
Additional	<empty>

and

Header	OPCODE=SQUERY, RESPONSE
Question	QNAME=SRI-NIC.ARPA., QCLASS=IN, QTYPE=*
Answer	SRI-NIC.ARPA. 1290 IN HINFO DEC-2060 TOPS20
Authority	<empty>
Additional	<empty>

Neither of these answers have AA set, so neither response comes from authoritative data. The different contents and different TTLs suggest that the two servers cached data at different times, and that the first server cached the response to a QTYPE=A query and the second cached the response to a HINFO query.

RFC-1034/5 Domain Name System: Scenario

QNAME=SRI-NIC.ARPA, QTYPE=MX

This type of query might be result from a mailer trying to look up routing information for the mail destination `HOSTMASTER@SRI-NIC.ARPA`. The response from `C.ISI.EDU` would be:

Header	OPCODE=SQUERY, RESPONSE, AA		
Question	QNAME=SRI-NIC.ARPA., QCLASS=IN, QTYPE=MX		
Answer	SRI-NIC.ARPA., 86400 IN	MX	0 SRI-NIC.ARPA.
Authority	<empty>		
Additional	SRI-NIC.ARPA. 86400 IN	A	26.0.0.73
		A	10.0.0.51

This response contains the MX RR in the answer section of the response. The additional section contains the address RRs because the name server at `C.ISI.EDU` guesses that the requester will need the addresses in order to properly use the information carried by the MX.

RFC-1034/5 Domain Name System: Scenario

QNAME=SRI-NIC.ARPA, QTYPE=NS

`C.ISI.EDU` would reply to this query with:

Header	OPCODE=SQUERY, RESPONSE, AA
Question	QNAME=SRI-NIC.ARPA., QCLASS=IN, QTYPE=NS
Answer	<empty>
Authority	<empty>
Additional	<empty>

The only difference between the response and the query is the AA and RESPONSE bits in the header. The interpretation of this response is that the server is authoritative for the name, and the name exists, but no RRs of type NS are present there.

RFC-1034/5 Domain Name System: Scenario

QNAME=SIR-NIC.ARPA, QTYPE=A

If a user mistyped a host name, we might see this type of query.

`C.ISI.EDU` would answer it with:

Header	OPCODE=SQUERY, RESPONSE, AA, RCODE=NE
Question	QNAME=SRI-NIC.ARPA., QCLASS=IN, QTYPE=A
Answer	<empty>
Authority	. SOA SRI-NIC.ARPA. HOSTMASTER.SRI-NIC.ARPA. 870611 1800 300 604800 86400
Additional	<empty>

This response states that the name does not exist. This condition is signalled in the response code (RCODE) section of the header.

The SOA RR in the authority section is the optional negative caching information which allows the resolver using this response to assume that the name will notexist for the SOA MINIMUM (86400) seconds.

RFC-1034/5 Domain Name System: Scenario

QNAME=BRL.MIL, QTYPE=A

If this query is sent to C.ISI.EDU, the reply would be:

Header	OPCODE=SQUERY, RESPONSE			
Question	QNAME=BRL.MIL, QCLASS=IN, QTYPE=A			
Answer	<empty>			
Authority	MIL.	86400 IN	NS	SRI-NIC.ARPA.
		86400	NS	A.ISI.EDU
Additional	A.ISI.EDU		A	26.3.0.103
	SRI-NIC.ARPA		A	26.0.0.73
			A	10.0.0.51

This response has an empty answer section, but is not authoritative, so it is a referral. The name server on C.ISI.EDU, realizing that it is not authoritativefor the MIL domain, has referred the requester to servers on A.ISI.EDU and SRI-NIC.ARPA, which it knows are authoritative for the MIL domain.

RFC-1034/5 Domain Name System: Scenario

QNAME=USC-ISIC.ARPA, QTYPE=A

The response to this query from A.ISI.EDU would be:

Header	OPCODE=SQUERY, RESPONSE, AA
Question	QNAME=USC-ISIC.ARPA., QCLASS=IN, QTYPE=A
Answer	USC-ISIC.ARPA. 86400 IN CNAME C.ISI.EDU. C.ISI.EDU 86400 IN A 10.0.0.52
Authority	<empty>
Additional	<empty>

Note that the AA bit in the header guarantees that the data matching QNAME is authoritative, but does not say anything about whether the data for C.ISI.EDU is authoritative. This complete reply is possible because A.ISI.EDU happens to be authoritative for both the ARPA domain where USC-ISIC.ARPA is found and the ISI.EDU domain where C.ISI.EDU data is found.

If the same query was sent to C.ISI.EDU, its response might be the same as shown above if it had its own address in its cache, but might also be:

Header	OPCODE=SQUERY, RESPONSE, AA			
Question	QNAME=USC-ISIC.ARPA., QCLASS=IN, QTYPE=A			
Answer	USC-ISIC.ARPA.	86400	IN CNAME	C.ISI.EDU.
Authority	ISI.EDU.	172800	IN NS	VAXA.ISI.EDU.
			NS	A.ISI.EDU.
			NS	VENERA.ISI.EDU.
Additional	VAXA.ISI.EDU.	172800	A	10.2.0.27
		172800	A	128.9.0.33
	VENERA.ISI.EDU.	172800	A	10.1.0.52
		172800	A	128.9.0.32
	A.ISI.EDU	172800	A	26.3.0.103

This reply contains an authoritative reply for the alias USC-ISIC.ARPA, plus a referral to the name servers for ISI.EDU. This sort of reply isn't very likely given that the query is for the host name of the name server being asked, but would be common for other aliases.

RFC-1034/5 Domain Name System: Scenario

QNAME=USC-ISIC.ARPA, QTYPE=CNAME

If this query is sent to either `A.ISI.EDU` or `C.ISI.EDU`, the reply would be:

Header	OPCODE=SQUERY
Question	QNAME=USC-ISIC.ARPA., QCLASS=IN, QTYPE=A
Answer	ISC=ISIC.ARPA. 86400 IN CNAME C.ISI.EDU.
Authority	<empty>
Additional	<empty>

Because QTYPE=CNAME, the CNAME RR itself answers the query, and the name server doesn't attempt to look up anything for `C.ISI.EDU`. (Except possibly for the additional section.)

Example Resolution

The following examples illustrate the operations a resolver must perform for its client. We assume that the resolver is starting without a cache, as might be the case after system boot. We further assume that the system is not one of the hosts in the data and that the host is located somewhere on net 26, and that its safety belt (SBELT) data structure has the following information:

```
Match count = -1
SRI-NIC.ARPA.   26.0.0.73       10.0.0.51
A.ISI.EDU.      26.3.0.103
```

This information specifies servers to try, their addresses, and a match count of -1, which says that the servers aren't very close to the target. Note that the -1 isn't supposed to be an accurate closeness measure, just a value so that later stages of the algorithm will work.

The following examples illustrate the use of a cache, so each example assumes that previous requests have completed.

RFC-1034/5 Domain Name System: Scenario

Resolve MX for ISI.EDU.

Suppose the first request to the resolver comes from the local mailer, which has mail for PVM@ISI.EDU. The mailer might then ask for type MX RRs for the domain name ISI.EDU.

The resolver would look in its cache for MX RRs at ISI.EDU, but the empty cache wouldn't be helpful. The resolver would recognize that it needed to query foreign servers and try to determine the best servers to query. This search would look for NS RRs for the domains ISI.EDU, EDU, and the root. These searches of the cache would also fail. As a last resort, the resolver would use the information from the SBELT, copying it into its SLIST structure.

At this point the resolver would need to pick one of the three available addresses to try. Given that the resolver is on net 26, it should choose either 26.0.0.73 or 26.3.0.103 as its first choice. It would then send off a query of the form:

Header	OPCODE=SQUERY
Question	QNAME=ISI.EDU., QCLASS=IN, QTYPE=MX
Answer	<empty>
Authority	<empty>
Additional	<empty>

The resolver would then wait for a response to its query or a timeout. If the timeout occurs, it would try different servers, then different addresses of the same servers, lastly retrying addresses already tried. It might eventually receive a reply from SRI-NIC.ARPA:

Header	OPCODE=SQUERY, RESPONSE			
Question	QNAME=ISI.EDU., QCLASS=IN, QTYPE=MX			
Answer	<empty>			
Authority	ISI.EDU.	172800 IN	NS	VAXA.ISI.EDU.
			NS	A.ISI.EDU.
			NS	VENERA.ISI.EDU.
Additional	VAXA.ISI.EDU.	172800	A	10.2.0.27
		172800	A	128.9.0.33
	VENERA.ISI.EDU.	172800	A	10.1.0.52
		172800	A	128.9.0.32
	A.ISI.EDU	172800	A	26.3.0.103

The resolver would notice that the information in the response gave a closer delegation to ISI.EDU than its existing SLIST (since it matches three labels). The resolver would then cache the information in this response and use it to set up a new SLIST:

```
Match count = 3
A.ISI.EDU.      26.3.0.103
VAXA.ISI.EDU.   10.2.0.27       128.9.0.33
VENERA.ISI.EDU. 10.1.0.52       128.9.0.32
```

A.ISI.EDU appears on this list as well as the previous one, but that is purely coincidental. The resolver would again start transmitting and waiting for responses. Eventually it would get an answer:

Header	OPCODE=SQUERY, RESPONSE, AA			
Question	QNAME=ISI.EDU., QCLASS=IN, QTYPE=MX			
Answer	ISI.EDU.		MX 10 VENERA.ISI.EDU. MX 20 VAXA.ISI.EDU.	
Authority	<empty>			
Additional	VAXA.ISI.EDU.	172800	A	10.2.0.27
		172800	A	128.9.0.33
	VENERA.ISI.EDU.	172800	A	10.1.0.52
		172800	A	128.9.0.32

The resolver would add this information to its cache, and return the MX RRs to its client.

RFC-1034/5 Domain Name System: Scenario

Get the Host Name for Address 26.6.0.65

The resolver would translate this into a request for PTR RRs for `65.0.6.26.IN-ADDR.ARPA`. This information is not in the cache, so the resolver would look for foreign servers to ask. No servers would match, so it would use SBELT again. (Note that the servers for the `ISI.EDU` domain are in the cache, but `ISI.EDU` is not an ancestor of `65.0.6.26.IN-ADDR.ARPA`, so the SBELT is used.)

Since this request is within the authoritative data of both servers in SBELT, eventually one would return:

Header	**OPCODE=SQUERY, RESPONSE, AA**
Question	**QNAME=65.0.6.26.IN-ADDR.ARPA.,QCLASS=IN,QTYPE=PTR**
Answer	**65.0.6.26.IN-ADDR.ARPA. PTR ACC.ARPA.**
Authority	**<empty>**
Additional	**<empty>**

RFC-1034/5 Domain Name System: Scenario

Get the Host Address of `poneria.ISI.EDU`

This request would translate into a type A request for `poneria.ISI.EDU`. The resolver would not find any cached data for this name, but would find the NS RRs in the cache for `ISI.EDU` when it looks for foreign servers to ask. Using this data, it would construct a SLIST of the form:

```
Match count = 3

A.ISI.EDU.      26.3.0.103
VAXA.ISI.EDU.   10.2.0.27       128.9.0.33
VENERA.ISI.EDU. 10.1.0.52
```

`A.ISI.EDU` is listed first on the assumption that the resolver orders its choices by preference, and `A.ISI.EDU` is on the same network.

One of these servers would answer the query.

[Dyer 87] Dyer, S., and F. Hsu, "Hesiod", Project Athena Technical Plan - Name Service, April 1987, version 1.9.

Describes the fundamentals of the Hesiod name service.

[IEN-116] J. Postel, "Internet Name Server", IEN-116, USC/Information Sciences Institute, August 1979.

A name service obsoleted by the Domain Name System, but still in use.

[Quarterman 86] Quarterman, J., and J. Hoskins, "Notable Computer Networks",Communications of the ACM, October 1986, volume 29, number 10.

RFC-1042 A Standard for the Transmission of IP Datagrams over IEEE 802 Networks

J. Postel & J. Reynolds
USC/Information Sciences Institute
February 1988

Status of this Memo

This RFC specifies a standard method of encapsulating the Internet Protocol (IP) **RFC-791** datagrams and Address Resolution Protocol (ARP) **RFC-826** requests and replies on IEEE 802 Networks. This RFC specifies a protocol standard for the Internet community. Distribution of this memo is unlimited.

Acknowledgment

This memo would not exist with out the very significant contributions of Drew Perkins of Carnegie Mellon University, Jacob Rekhter of the T.J. Watson Research Center, IBM Corporation, and Joseph Cimmino of the University of Maryland.

RFC-1042 IP and ARP on IEEE 802 Networks

Introduction

The goal of this specification is to allow compatible and interoperable implementations for transmitting IP datagrams and ARP requests and replies. To achieve this it may be necessary in a few cases to limit the use that IP and ARP make of the capabilities of a particular IEEE 802 standard.

The IEEE 802 specifications define a family of standards for Local Area Networks (LANs) that deal with the Physical and Data Link Layers as defined by the ISO Open System Interconnection Reference Model (ISO/OSI). Several Physical Layer standards (802.3, 802.4, and 802.5) [3,4,5] and one Data Link Layer Standard (802.2) [6] have been defined. The IEEE Physical Layer standards specify the ISO/OSI Physical Layer and the Media Access Control Sublayer of the ISO/OSI Data Link Layer. The 802.2 Data Link Layer standard specifies the Logical Link Control Sublayer of the ISO/OSI Data Link Layer.

This memo describes the use of IP and ARP on the three types of networks. At this time, it is not necessary that the use of IP and ARP be consistent across all three types of networks, only that it be consistent within each type. This may change in the future as new IEEE 802 standards are defined and the existing standards are revised allowing for interoperability at the Data Link Layer.

It is the goal of this memo to specify enough about the use of IP and ARP on each type of network to ensure that:

(1) all equipment using IP or ARP on 802.3 networks will interoperate,

(2) all equipment using IP or ARP on 802.4 networks will interoperate,

(3) all equipment using IP or ARP on 802.5 networks will interoperate.

Of course, the goal of IP is interoperability between computers attached to different networks, when those networks are interconnected via an IP gateway [8]. The use of IEEE 802.1 compatible Transparent Bridges to allow interoperability across different networks is not fully described pending completion of that standard.

RFC-1042 IP and ARP on IEEE 802 Networks

Description

IEEE 802 networks may be used as IP networks of any class (A, B, or C). These systems use two Link Service Access Point (LSAP) fields of the LLC header in much the same way the ARPANET uses the "link" field. Further, there is an extension of the LLC header called the Sub-Network Access Protocol (SNAP).

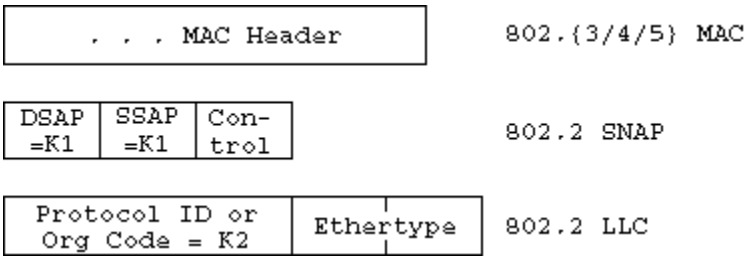
IP datagrams are sent on IEEE 802 networks encapsulated within the 802.2 LLC and SNAP data link layers, and the 802.3, 802.4, or 802.5 physical networks layers. The SNAP is used with an Organization Code indicating that the following 16 bits specify the EtherType code.

Normally, all communication is performed using 802.2 type 1 communication. Consenting systems on the same IEEE 802 network may use 802.2 type 2 communication after verifying that it is supported by both nodes. This is accomplished using the 802.2 XID mechanism. However, type 1 communication is the recommended method at this time and must be supported by all implementations. The rest of this specification assumes the use of type 1 communication.

The IEEE 802 networks may have 16-bit or 48-bit physical addresses. This specification allows the use of either size of address within a given IEEE 802 network.

Note that the 802.3 standard specifies a transmission rate of from 1 to 20 megabit/second, the 802.4 standard specifies 1, 5, and 10 megabit/second, and the 802.5 standard specifies 1 and 4 megabit/second. The typical transmission rates used are 10 megabit/second for 802.3, 10 megabit/second for 802.4, and 4 megabit/second for 802.5. However, this specification for the transmission of IP Datagrams does not depend on the transmission rate.

Header Format



The total length of the LLC Header and the SNAP header is 8-octets, making the 802.2 protocol overhead come out on an nice boundary.

The K1 value is 170 (decimal).

The K2 value is 0 (zero).

The control value is 3 (Unnumbered Information).

RFC-1042 IP and ARP on IEEE 802 Networks

Address Mappings

The mapping of 32-bit Internet addresses to 16-bit or 48-bit IEEE 802 addresses must be done via the dynamic discovery procedure of the Address Resolution Protocol (ARP) RFC-826.

Internet addresses are assigned arbitrarily on Internet networks. Each host's implementation must know its own Internet address and respond to Address Resolution requests appropriately. It must also use ARP to translate Internet addresses to IEEE 802 addresses when needed.

The ARP Details

The ARP protocol has several fields that parameterize its use in any specific context. These fields are:

hrd	16 - bits	The Hardware Type Code
pro	16 - bits	The Protocol Type Code
hln	8 - bits	Octets in each hardware address
pln	8 - bits	Octets in each protocol address
op	16 - bits	Operation Code

The hardware type code assigned for the IEEE 802 networks (of all kinds) is 6.

The protocol type code for IP is 2048.

The hardware address length is 2 for 16-bit IEEE 802 addresses, or 6 for 48-bit IEEE 802 addresses.

The protocol address length (for IP) is 4.

The operation code is 1 for request and 2 for reply.

Broadcast Address

The broadcast Internet address (the address on that network with a host part of all binary ones) should be mapped to the broadcast IEEE 802 address (of all binary ones).

Trailer Formats

Some versions of Unix 4.x bsd use a different encapsulation method in order to get better network performance with the VAX virtual memory architecture. Consenting systems on the same IEEE 802 network may use this format between themselves. Details of the trailer encapsulation method may be found in RFC-893. However, all hosts must be able to communicate using the standard (non-trailer) method.

Byte Order

As described in Appendix B of the Internet Protocol specification RFC-791, the IP datagram is transmitted over IEEE 802 networks as a series of 8-bit bytes. This byte transmission order has been called "big-endian" [11].

RFC-1042 IP and ARP on IEEE 802 Networks

Maximum Transmission Unit

The Maximum Transmission Unit (MTU) differs on the different types of IEEE 802 networks. In the following there are comments on the MTU for each type of IEEE 802 network. However, on any particular network all hosts must use the same MTU. In the following, the terms "maximum packet size" and "maximum transmission unit" are equivalent.

RFC-1042 IP on IEEE 802 Networks - Frame Format and MAC Level Issues

For all hardware types

IP datagrams and ARP requests and replies are transmitted in standard 802.2 LLC Type 1 Unnumbered Information format, control code 3, with the DSAP and the SSAP fields of the 802.2 header set to 170, the assigned global SAP value for SNAP [6]. The 24-bit Organization Code in the SNAP is zero, and the remaining 16 bits are the EtherType from Assigned Numbers (IP = 2048, ARP = 2054).

IEEE 802 packets may have a minimum size restriction. When necessary, the data field should be padded (with octets of zero) to meet the IEEE 802 minimum frame size requirements. This padding is not part of the IP datagram and is not included in the total length field of the IP header.

For compatibility (and common sense) the minimum packet size used with IP datagrams is 28 octets, which is 20 (minimum IP header) + 8 (LLC+SNAP header) = 28 octets (not including the MAC header).

The minimum packet size used with ARP is 24 octets, which is 20 (ARP with 2 octet hardware addresses and 4 octet protocol addresses) + 8 (LLC+SNAP header) = 24 octets (not including the MAC header).

In typical situations, the packet size used with ARP is 32 octets, which is 28 (ARP with 6 octet hardware addresses and 4 octet protocol addresses) + 8 (LLC+SNAP header) = 32 octets (not including the MAC header).

IEEE 802 packets may have a maximum size restriction. Implementations are encouraged to support full-length packets.

For compatibility purposes, the maximum packet size used with IP datagrams or ARP requests and replies must be consistent on a particular network.

Gateway implementations must be prepared to accept full-length packets and fragment them when necessary.

Host implementations should be prepared to accept full-length packets, however hosts must not send datagrams longer than 576 octets unless they have explicit knowledge that the destination is prepared to accept them. A host may communicate its size preference in TCP based applications via the TCP Maximum Segment Size option [RFC-879].

Datagrams on IEEE 802 networks may be longer than the general Internet default maximum packet size of 576 octets. Hosts connected to an IEEE 802 network should keep this in mind when sending datagrams to hosts not on the same IEEE 802 network. It may be appropriate to send smaller datagrams to avoid unnecessary fragmentation at intermediate gateways. Please see "TCP Maximum Segment Size and Related Information" [RFC-879] for further information.

IEEE 802.2 Details

While not necessary for supporting IP and ARP, all implementations are required to support IEEE 802.2 standard Class I service. This requires supporting Unnumbered Information (UI) Commands, eXchange IDentification (XID) Commands and Responses, and TEST link (TEST) Commands and Responses.

When either an XID or a TEST command is received a response must be returned; with the Destination and Source addresses, and the DSAP and SSAP swapped.

When responding to an XID or a TEST command the sense of the poll/final bit must be preserved. That is, a command received with the poll/final bit reset must have the response returned with the poll/final bit reset and vice versa.

The XID command or response has an LLC control field value of 175 (decimal) if poll is off or 191 (decimal) if poll is on. (See Appendix on Numbers.)

The TEST command or response has an LLC control field value of 227 (decimal) if poll is off or 243 (decimal) if poll is on. (See Appendix on Numbers.)

A command frame is identified with high order bit of the SSAP address reset. Response frames have high order bit of the SSAP address set to one.

XID response frames should include an 802.2 XID Information field of 129.1.0 indicating Class I (connectionless) service. (type 1).

TEST response frames should echo the information field received in the corresponding TEST command frame.

IEEE 802.3 Details

A particular implementation of an IEEE 802.3 Physical Layer is denoted using a three field notation. The three fields are data rate in megabit/second, medium type, and maximum segment length in hundreds of meters. One combination of of 802.3 parameters is 10BASE5 which specifies a 10 megabit/second transmission rate, baseband medium, and 500 meter segments. This correspondes to the specifications of the familiar "Ethernet" network.

The MAC header contains 6 (2) octets of source address, 6 (2) octets of destination address, and 2 octets of length. The MAC trailer contains 4 octets of Frame Check Sequence (FCS), for a total of 18 (10) octets.

IEEE 802.3 networks have a minimum packet size that depends on the transmission rate. For type 10BASE5 802.3 networks the minimum packet size is 64 octets.

IEEE 802.3 networks have a maximum packet size which depends on the transmission rate. For type 10BASE5 802.3 networks the maximum packet size is 1518 octets including all octets between the destination address and the FCS inclusive.

This allows 1518 - 18 (MAC header+trailer) - 8 (LLC+SNAP header) = 1492 for the IP datagram (including the IP header). Note that 1492 is not equal to 1500 which is the MTU for Ethernet networks.

IEEE 802.4 Details

The MAC header contains 1 octet of frame control, 6 (2) octets of source address, and 6 (2) octets of destination address. The MAC trailer contains 4 octets of Frame Check Sequence (FCS), for a total of 17 (9) octets.

IEEE 802.4 networks have no minimum packet size.

IEEE 802.4 networks have a maximum packet size of 8191 octets including all octets between the frame control and the FCS inclusive.

This allows 8191 - 17 (MAC header+trailer) - 8 (LLC+SNAP header) = 8166 for the IP datagram (including the IP header).

IEEE 802.5 Details

The current standard for token ring's, IEEE 802.5-1985, specifies the operation of single ring networks. However, most implementations of 802.5 have added extensions for multi-ring networks using source-routing of packets at the MAC layer. There is now a Draft Addendum to IEEE 802.5, "Enhancement for Multi-Ring Networks" which attempts to standardize these extensions. Unfortunately, the most recent draft (November 10, 1987) is still rapidly evolving. More importantly, it differs significantly from the existing implementations. Therefore, the existing implementations of 802.5 [13] are described but no attempt is made to specify any future standard.

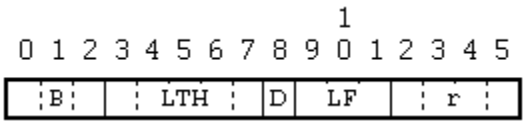
The MAC header contains 1 octet of access control, 1 octet of frame control, 6 (2) octets of source address, 6 (2) octets of destination address, and (for multi-ring networks) 0 to 18 octets of Routing Information Field (RIF). The MAC trailer contains 4 octets of FCS, for a total of 18 (10) to 36 (28) octets. There is one additional octet of frame status after the FCS.

RFC-1042 IP on IEEE 802 Networks - 802.5 Frame Format and MAC Level Issues

Multi-Ring Extension Details

The presence of a Routing Information Field is indicated by the Most Significant Bit (MSB) of the source address, called the Routing Information Indicator (RII). If the RII equals zero, a RIF is not present. If the RII equals 1, the RIF is present. Although the RII is indicated in the source address, it is not part of a stations MAC layer address. In particular, the MSB of a destination address is the individual/group address indicator, and if set will cause such frames to be interpreted as multicasts. Implementations should be careful to reset the RII to zero before passing source addresses to other protocol layers which may be confused by their presence.

The RIF consists of a two-octet Routing Control (RC) field followed by 0 to 8 two-octet Route-Designator (RD) fields. The RC for all-routes broadcast frames is formatted as follows:



B - Broadcast Indicators: 3 bits

The Broadcast Indicators are used to indicate the routing desired for a particular frame. A frame may be routed through a single specified route, through every distinct non-repeating route in a multi-ring network, or through a single route determined by a spanning tree algorithm such that the frame appears on every ring exactly once. The values which may be used at this time are (in binary):

000 - Non-broadcast (specific route)
100 - All-routes broadcast (global broadcast)
110 - Single-route broadcast (limited broadcast)

All other values are reserved for future use.

LTH - Length: 5 bits

The Length bits are used to indicate the length or the RI field, including the RC and RD fields. Only even values between 2 and 30 inclusive are allowed.

D - Direction Bit: 1 bit

The D bit specifies the order of the RD fields. If D equals 1, the routing-designator fields are specified in reverse order.

LF - Largest Frame: 3 bits

The LF bits specify the maximum MTU supported by all bridges along a specific route. All multi-ring broadcast frames should be transmitted with a value at least as large as the supported MTU. The values used are:

LF (binary)	MAC MTU	IP MTU
000	552	508
001	1064	1020
010	2088	2044
011	4136	4092
100	8232	8188

All other values are reserved for future use.

The receiver should compare the LF received with the MTU. If the LF is greater than or equal to the MTU then no action is taken; however, if the LF is less than the MTU the frame is rejected.

There are actually three possible actions if LF < MTU. First is the one required for this specification (reject the frame). Second is to reduce the MTU for all hosts to equal the LF. And, third is to keep a separate MTU per communicating host based on the received LFs.

r - reserved: 4 bits

These bits are reserved for future use and must be set to 0 by the transmitter and ignored by the receiver.

IEEE 802.5 Packet Size Issues

It is not necessary for an implementation to interpret routing-designators. Their format is left unspecified. Routing-designators should be transmitted exactly as received.

IEEE 802.5 networks have no minimum packet size.

IEEE 802.5 networks have a maximum packet size based on the maximum time a node may hold the token. This time depends on many factors including the data signalling rate and the number of nodes on the ring. The determination of maximum packet size becomes even more complex when multi-ring networks with bridges are considered.

Given a token-holding time of 9 milliseconds and a 4 megabit/second ring, the maximum packet size possible is 4508 octets including all octets between the access control and the FCS inclusive.

This allows 4508 - 36 (MAC header+trailer with 18 octet RIF) - 8 (LLC+SNAP header) = 4464 for the IP datagram (including the IP header).

However, some current implementations are known to limit packets to 2046 octets (allowing 2002 octets for IP). It is recommended that all implementations support IP packets of at least 2002 octets.

By convention, source routing bridges used in multi-ring 802.5 networks will not support packets larger than 8232 octets. With a MAC header+trailer of 36 octets and the LLC+SNAP header of 8 octets, the IP datagram (including IP header) may not exceed 8188 octets.

A source routing bridge linking two rings may be configured to limit the size of packets forwarded to 552 octets, with a MAC header+trailer of 36 octets and the LLC+SNAP of 8 octets, the IP datagram (including the IP header) may be limited to 508 octets. This is less that the default IP MTU of 576 octets, and may cause significant performance problems due to excessive datagram fragmentation. An implementation is not required to support an MTU of less than 576 octets, although it is suggest that the MTU be a user-configurable parameter to allow for it.

IEEE 802.5 Broadcast Issues

IEEE 802.5 networks support three different types of broadcasts. All-Stations broadcasts are sent with no RIF or with the Broadcast Indicators set to 0 and no Routing Designators, and are copied once by all stations on the local ring. All-Routes broadcasts are sent with the corresponding Broadcast Indicators and result in multiple copies equal to the number of distinct non-repeating routes a packet may follow to a particular ring. Single-Route broadcasts result in exactly one copy of a frame being received by all stations on the multi-ring network.

The dynamic address discovery procedure is to broadcast an ARP request. To limit the number of all rings broadcasts to a minimum, it is desirable (though not required) that an ARP request first be sent as an all-stations broadcast, without a Routing Information Field (RIF). If the all-stations (local ring) broadcast is not supported or if the all-stations broadcast is unsuccessful after some reasonable time has elapsed, then send the ARP request as an all-routes or single-route broadcast with an empty RIF (no routing designators). An all-routes broadcast is preferable since it yields an amount of fault tolerance. In an environment with multiple redundant bridges, all-routes broadcast allows operation in spite of spanning-tree bridge failures. However, single-route broadcasts may be used if IP and ARP must use the same broadcast method.

When an ARP request or reply is received, all implementations are required to understand frames with no RIF (local ring) and frames with an empty RIF (also from the local ring). If the implementation supports multi-ring source routing, then a non- empty RIF is stored for future transmissions to the host originating the ARP request or reply. If source routing is not supported them all packets with non-empty RIFs should be gracefully ignored. This policy will allow all implementations in a single ring environment, to interoperate, whether or not they support the multi-ring extensions.

It is possible that when sending an ARP request via an all-routes broadcast that multiple copies of the request will arrive at the destination as a result of the request being forwarded by several bridges. However, these "copies" will have taken different routes so the contents of the RIF will differ. An implementation of ARP in this context must determine which of these "copies" to use and to ignore the others. There are three obvious and legal strategies: (1) take the first and ignore the rest (that is, once you have an entry in the ARP cache don't change it), (2) take the last, (that is, always up date the ARP cache with the latest ARP message), or (3) take the one with the shortest path, (that is, replace the ARP cache information with the latest ARP message data if it is a shorter route). Since there is no problem of incompatibility for interworking of different implementations if different strategies are chosen, the choice is up to each implementor. The recipient of the ARP request must send an ARP reply as a point to point message using the RIF information.

The RIF information should be kept distinct from the ARP table. That is, there is, in principle, the ARP table to map from IP addresses to 802 48-bit addresses, and the RIF table to map from those to 802.5 source routes, if necessary. In practical implementations it may be convenient to store the ARP and RIF information together.

> Storing the information together may speed up access to the information when it is used. On the other hand, in a generalized implementation for all types of 802 networks a significant amount of memory might be wasted in an ARP cache if space for the RIF information were always reserved.

IP broadcasts (datagrams with a IP broadcast address) must be sent as 802.5 single-route broadcasts. Unlike ARP, all-routes broadcasts are not desirable for IP. Receiving multiple copies of IP broadcasts would have undesirable effects on many protocols using IP. As with

ARP, when an IP packet is received, all implementations are required to understand frames with no RIF and frames with an empty RIF.

Since current interface hardware allows only one group address, and since the functional addresses are not globally unique, IP and ARP do not use either of these features. Further, in the IBM style 802.5 networks there are only 31 functional addresses available for user definition.

IP precedence should not be mapped to 802.5 priority. All IP and ARP packets should be sent at the default 802.5 priority. The default priority is 3.

After packet transmission, 802.5 provides frame not copied and address not recognized indicators. Implementations may use these indicators to provide some amount of error detection and correction. If the frame not copied bit is set but the address not recognized bit is reset, receiver congestion has occurred. It is suggested, though not required, that hosts should retransmit the offending packet a small number of times (4) or until congestion no longer occurs. If the address not recognized bit is set, an implementation has 3 options: (1) ignore the error and throw the packet away, (2) return an ICMP destination unreachable message to the source, or (3) delete the ARP entry which was used to send this packet and send a new ARP request to the destination address. The latter option is the preferred approach since it will allow graceful recovery from first hop bridge and router failures and changed hardware addresses.

Interoperation with Ethernet

It is possible to use the Ethernet link level protocol [12] on the same physical cable with the IEEE 802.3 link level protocol. A computer interfaced to a physical cable used in this way could potentially read both Ethernet and 802.3 packets from the network. If a computer does read both types of packets, it must keep track of which link protocol was used with each other computer on the network and use the proper link protocol when sending packets.

One should note that in such an environment, link level broadcast packets will not reach all the computers attached to the network, but only those using the link level protocol used for the broadcast.

Since it must be assumed that most computers will read and send using only one type of link protocol, it is recommended that if such an environment (a network with both link protocols) is necessary, an IP gateway be used as if there were two distinct networks.

Note that the MTU for the Ethernet allows a 1500 octet IP datagram, with the MTU for the 802.3 network allows only a 1492 octet IP datagram.

Appendix on Numbers

The IEEE likes to specify numbers in bit transmission order, or bit- wise little-endian order. The Internet protocols are documented in byte-wise big-endian order. This may cause some confusion about the proper values to use for numbers. Here are the conversions for some numbers of interest.

Number	IEEE HEX	IEEE Binary	Internet Binary	Internet Decimal
UI Op Code	C0	11000000	00000011	3
SAP for SNAP	55	01010101	10101010	170
XID	F5	11110101	10101111	175
XID	FD	11111101	10111111	191
TEST	C7	11000111	11100011	227
TEST	CF	11001111	11110011	243
Info	818000			129.1.0

References

[1] Postel, J., "Internet Protocol", RFC-791, USC/Information Sciences Institute, September 1981.

[2] Plummer, D., "An Ethernet Address Resolution Protocol - or - Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware", RFC-826, MIT, November 1982.

[3] IEEE, "IEEE Standards for Local Area Networks: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications", IEEE, New York, New York, 1985.

[4] IEEE, "IEEE Standards for Local Area Networks: Token-Passing Bus Access Method and Physical Layer Specification", IEEE, New York, New York, 1985.

[5] IEEE, "IEEE Standards for Local Area Networks: Token Ring Access Method and Physical Layer Specifications", IEEE, New York, New York, 1985.

[6] IEEE, "IEEE Standards for Local Area Networks: Logical Link Control", IEEE, New York, New York, 1985.

[7] Reynolds, J.K., and J. Postel, "Assigned Numbers", RFC-1010, USC/Information Sciences Institute, May 1987.

[8] Braden, R., and J. Postel, "Requirements for Internet Gateways", RFC-1009, USC/Information Sciences Institute, June 1987.

[9] Leffler, S., and M. Karels, "Trailer Encapsulations", RFC-893, University of California at Berkeley, April 1984.

[10] Postel, J., "The TCP Maximum Segment Size Option and Related

Topics", RFC-879, USC/Information Sciences Institute, November 1983.

[11] Cohen, D., "On Holy Wars and a Plea for Peace", Computer, IEEE, October 1981.

[12] D-I-X, "The Ethernet - A Local Area Network: Data Link Layer and Physical Layer Specifications", Digital, Intel, and Xerox, November 1982.

[13] IBM, "Token-Ring Network Architecture Reference", Second Edition, SC30-3374-01, August 1987.

RFC-1049 A Content-Type Header Field for Internet Messages

M. Sirbu; CMU
March 1988

Status Of This Memo

This RFC suggests proposed additions to the Internet Mail Protocol, RFC-822, for the Internet community, and requests discussion and suggestions for improvements. Distribution of this memo is unlimited. As of May 1990 the IAB has classified this RFC as **recommended**.

Abstract

A standardized Content-type field allows mail reading systems to automatically identify the type of a structured message body and to process it for display accordingly. The structured message body must still conform to the RFC-822 requirements concerning allowable characters. A mail reading system need not take any specific action upon receiving a message with a valid Content-Type header field. The ability to recognize this field and invoke the appropriate display process accordingly will, however, improve the readability of messages, and allow the exchange of messages containing mathematical symbols, or foreign language characters.

RFC-1049 A Content-Type Header Field

Introduction

As defined in RFC-822, [RFC-822], an electronic mail message consists of a number of defined header fields, some containing structured information (e.g., date, addresses), and a message body consisting of an unstructured string of ASCII characters.

The success of the Internet mail system has led to a desire to use the mail system for sending around information with a greater degree of structure, while remaining within the constraints imposed by the limited character set. A prime example is the use of mail to send a document with embedded TROFF formatting commands. A more sophisticated example would be a message body encoded in a Page Description Language (PDL) such as Postscript. In both cases, simply mapping the ASCII characters to the screen or printer in the usual fashion will not render the document image intended by the sender; an additional processing step is required to produce an image of the message text on a display device or a piece of paper.

In both of these examples, the message body contains only the legal character set, but the content has a structure which produces some desirable result after appropriate processing by the recipient. If a message header field could be used to indicate the structuring technique used in the message body, then a sophisticated mail system could use such a field to automatically invoke the appropriate processing of the message body. For example, a header field which indicated that the message body was encoded using Postscript could be used to direct a mail system running under Sun Microsystem's NEWS window manager to process the Postscript to produce the appropriate page image on the screen.

Private header fields (beginning with "X-") are already being used by some systems to affect such a result (e.g., the Andrew Message System developed at Carnegie Mellon University). However, the widespread use of such techniques will require general agreement on the name and allowed parameter values for a header field to be used for this purpose.

We propose that a new header field, "Content-type:" be recognized as the standard field for indicating the structure of the message body. The contents of the "Content-Type:" field are parameters which specify what type of structure is used in the message body.

Note that we are not proposing that the message body contain anything other than ASCII characters as specified in RFC-822. Whatever structuring is contained in the message body must be represented using only the allowed ASCII characters. Thus, this proposal should have no impact on existing mailers, only on mail reading systems.

At the same time, this restriction eliminates the use of more general structuring techniques such as Abstract Syntax Notation, (CCITT Recommendation X.409) as used in the X.400 messaging standard, which are octet-oriented.

This is not the first proposal for structuring message bodies. RFC-767 [RFC-767] discusses a proposed technique for structuring multi-media mail messages. We are also aware that many users already employ mail to send TROFF, SCRIBE, TEX, Postscript or other structured information. Such postprocessing as is required must be invoked manually by the message recipient who looks at the message text displayed as conventional ASCII and recognizes that it is structured in some way that requires additional processing to be properly rendered. Our proposal is designed to facilitate automatic processing of messages by a mail reading system.

Problems with Structured Messages

Once we introduce the notion that a message body might require some processing other than simply painting the characters to the screen we raise a number of fundamental questions. These generally arise due to the certainty that some receiving systems will have the facilities to process the received message and some will not. The problem is what to do in the presence of systems with different levels of capability.

First, we must recognize that the purpose of structured messages is to be able to send types of information, ultimately intended for human consumption, not expressable in plain ASCII. Thus, there is no way in plain ASCII to send the italics, boldface, or greek characters that can be expressed in Postscript. If some different processing is necessary to render these glyphs, then that is the minimum price to be paid in order to send them at all.

Second, by insisting that the message body contain only ASCII, we insure that it will not "break" current mail reading systems which are not equipped to process the structure; the result on the screen may not be readily interpretable by the human reader, however.

If a message sender knows that the recipient cannot process Postscript, he or she may prefer that the message be revised to eliminate the use of italics and boldface, rather than appear incomprehensible. If Postscript is being used because the message contains passages in Greek, there may be no suitable ASCII equivalent, however.

Ideally, the details of structuring the message (or not) to conform to the capabilities of the recipient system could be completely hidden from the message sender. The distributed Internet mail system would somehow determine the capabilities of the recipient system, and convert the message automatically; or, if there was no way to send Greek text in ASCII, inform the sender that his message could not be transmitted.

In practice, this is a difficult task. There are three possible approaches:

1. Each mail system maintains a database of capabilities of remote systems it knows how to send to. Such a database would be very difficult to keep up to date.
2. The mail transport service negotiates with the receiving system as to its capabilities. If the receiving system cannot support the specified content type, the mail is transformed into conventional ASCII before transmission. This would require changes to all existing SMTP implementations, and could not be implemented in the case where RFC-822 type messages are being forwarded via Bitnet or other networks which do not implement SMTP.
3. An expanded directory service maintains information on mail processing capabilities of receiving hosts. This eliminates the need for real-time negotiation with the final destination, but still requires direct interaction with the directory service. Since directory querying is part of mail sending as opposed to mail composing/reading systems, this requires changes to existing mailers as well as a major change to the domain name directory service.

We note in passing that the X.400 protocol implements approach number 2, and that the Draft Recommendations for X.DS, the Directory Service, would support option 3.

In the interest of facilitating early usage of structured messages, we choose not to recommend any of the three approaches described above at the present time. In a forthcoming RFC we will propose a solution based on option 2, requiring modification to

mailers to support negotiation over capabilities. For the present, then, users would be obliged to keep their own private list of capabilities of recipients and to take care that they do not send Postscript, TROFF or other structured messages to recipients who cannot process them. The penalty for failure to do so will be the frustration of the recipient in trying to read a raw Postscript or TROFF file painted on his or her screen. Some System Administrators may attempt to implement option 1 for the benefit of their users, but this does not impose a requirement for changes on any other mail system.

We recognize that the long-term solution must require changes to mailers. However, in order to begin now to standardize the header fields, and to facilitate experimentation, we issue the present RFC.

RFC-1049 A Content-Type Header Field

Format of the Content-type Header Field

Whatever structuring technique is specified by the Content-type field, it must be known precisely to both the sender and the recipient of the message in order for the message to be properly interpreted. In general, this means that the allowed parameter values for the Content-type: field must identify a well-defined, standardized, document structuring technique. We do not preclude, however, the use of a Content-type: parameter value to specify a private structuring technique known only to the sender and the recipient.

More precisely, we propose that the Content-type: header field consist of up to four parameter values. The first, or type parameter names the structuring technique; the second, optional, parameter is a version number, ver-num, which indicates a particular version or revision of the standardized structuring technique. The third parameter is a resource reference, resource-ref, which may indicate a standard database of information to be used in interpreting the structured document. The last parameter is a comment.

In the Extended Backus Naur Form of RFC-822, we have:

Content-Type:= type [";" ver-num [";" 1#resource-ref]] [comment]

Type Values

Initially, the type parameter would be limited to the following set of values:

type:= "POSTSCRIPT"/"SCRIBE"/"SGML"/"TEX"/"TROFF"/ "DVI"/"X-"atom

These values are not case sensitive. POSTSCRIPT, Postscript, and POStscriPT are all equivalent.

POSTSCRIPT	Indicates the enclosed document consists of information encoded using the Postscript Page Definition Language developed by Adobe Systems, Inc. [1]
SCRIBE	Indicates the document contains embedded formatting information according to the syntax used by the Scribe document formatting language distributed by the Unilogic Corporation. [6]
SGML	Indicates the document contains structuring information to according the rules specified for the Standard Generalized Markup Language, IS 8879, as published by the International Organization for Standardization. [3] Documents structured according to the ISO DIS 8613--Office Docment Architecture and Interchange Format--may also be encoded using SGML syntax.
TEX	Indicates the document contains embedded formatting information according to the syntax of the TEX document production language. [4]
TROFF	Indicates the document contains embedded formatting information according to the syntax specified for the TROFF formatting package developed by AT&T Bell Laboratories. [5]
DVI	Indicates the document contains information according to the device independent file format produced by TROFF or TEX.
"X-"atom	Any type value beginning with the characters "X-" is a private value.

Version Number

Since standard structuring techniques in fact evolve over time, we leave room for specifying a version number for the content type. Valid values will depend upon the type parameter.

ver-num:= local-part

In particular, we have the following valid values:

For type=POSTSCRIPT

ver-num:= "1.0"/"2.0"/"null"

For type=SCRIBE

ver-num:= "3"/"4"/"5"/"null"

For type=SGML

ver-num:="IS.8879.1986"/"null"

Resource Reference

resource-ref:= local-part

As Apple has demonstrated with their implementation of the Laserwriter, a very general document structuring technique can be made more efficient by defining a set of macros or other similar resources to be used in interpreting any transmitted stream. The Macintosh transmits a LaserPrep file to the Laserwriter containing font and macro definitions which can be called upon by subsequent documents. The result is that documents as sent to the Laserwriter are considerably more compact than if they had to include the LaserPrep file each time. The Resource Reference parameter allows specification of a well known resource, such as a LaserPrep file, which should be used by the receiving system when processing the message.

Resource references could also include macro packages for use with TEX or references to preprocessors such as eqn and tbl for use with troff. Allowed values will vary according to the type parameter.

In particular, we propose the following values:

For type = POSTSCRIPT

resource-ref:= "laserprep2.9"/"laserprep3.0"/"laserprep3.1"/ "laserprep4.0"/local-part

For type = TROFF

resource-ref:= "eqn"/"tbl"/"me"/local-part

Comment

The comment field can be any additional comment text the user desires. Comments are enclosed in parentheses as specified in RFC-822.

Conclusion

A standardized Content-type field allows mail reading systems to automatically identify the type of a structured message body and to process it for display accordingly. The strcutured message body must still conform to the RFC-822 requirements concerning allowable characters. A mail reading system need not take any specific action upon receiving a message with valid Content-Type header field. The ability to recognize this field and invoke the appropriate display process accordingly will, however, improve the readability of messages, and allow the exchange of messages containing mathematical symbols, or foreign language characters. In the near term, the major use of a Content-Type: header field is likely to be for designating the message body as containing a Page Definition Language representation such as Postscript.

Additional type values shall be registered with Internet Assigned Numbers Coordinator at USC-ISI. Please contact:

Joyce K. Reynolds
USC Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292-6695
213-822-1511 JKReynolds@ISI.EDU

REFERENCES

1. Adobe Systems, Inc. Postscript Language Reference Manual. Addison-Wesley, Reading, Mass., 1985.

2. Crocker, David H. RFC-822: Standard for the Format of ARPA Internet Text Messages. Network Information Center, August 13, 1982.

3. ISO TC97/SC18. Standard Generalized Markup Language. Tech. Rept. DIS 8879, ISO, 1986.

4. Knuth, Donald E. The TEXbook. Addison-Wesley, Reading, Mass., 1984.

5. Ossanna, Joseph F. NROFF/TROFF User's Manual. Bell Laboratories, Murray Hill, New Jersey, 1976. Computing Science Technical Report No.54.

6. Unilogic. SCRIBE Document Production Software. Unilogic, 1985. Fourth Edition.

RFC-1055 A Non-Standard for the Transmission of IP Datagrams over Serial Lines SLIP

John Romkey
June 1988

Introduction

The TCP/IP protocol family runs over a variety of network media: IEEE 802.3 (ethernet) and 802.5 (token ring) LAN's, X.25 lines, satellite links, and serial lines. There are standard encapsulations for IP packets defined for many of these networks, but there is no standard for serial lines. SLIP, Serial Line IP, is a currently a de facto standard, commonly used for point-to-point serial connections running TCP/IP. It is not an Internet standard.

Distribution of this memo is unlimited.

RFC-1055 Transmission of IP Datagrams over Serial Lines (SLIP)

History

SLIP has its origins in the 3COM UNET TCP/IP implementation from the early 1980's. It is merely a packet framing protocol: SLIP defines a sequence of characters that frame IP packets on a serial line, and nothing more. It provides no addressing, packet type identification, error detection/correction or compression mechanisms. Because the protocol does so little, though, it is usually very easy to implement.

Around 1984, Rick Adams implemented SLIP for 4.2 Berkeley Unix and Sun Microsystems workstations and released it to the world. It quickly caught on as an easy reliable way to connect TCP/IP hosts and routers with serial lines.

SLIP is commonly used on dedicated serial links and sometimes for dialup purposes, and is usually used with line speeds between 1200bps and 19.2Kbps. It is useful for allowing mixes of hosts and routers to communicate with one another (host-host, host-router and router- router are all common SLIP network configurations).

Availability

SLIP is available for most Berkeley UNIX-based systems. It is included in the standard 4.3BSD release from Berkeley. SLIP is available for Ultrix, Sun UNIX and most other Berkeley-derived UNIX systems. Some terminal concentrators and IBM PC implementations also support it.

SLIP for Berkeley UNIX is available via anonymous FTP from uunet.uu.net in pub/sl.shar.Z. Be sure to transfer the file in binary mode and then run it through the UNIX uncompress program. Take the resulting file and use it as a shell script for the UNIX /bin/sh (for instance, /bin/sh sl.shar).

RFC-1055 Transmission of IP Datagrams over Serial Lines (SLIP)

Protocol

The SLIP protocol defines two special characters: END and ESC. END is octal 300 (decimal 192) and ESC is octal 333 (decimal 219) not to be confused with the ASCII ESCape character; for the purposes of this discussion, ESC will indicate the SLIP ESC character. To send a packet, a SLIP host simply starts sending the data in the packet. If a data byte is the same code as END character, a two byte sequence of ESC and octal 334 (decimal 220) is sent instead. If it the same as an ESC character, an two byte sequence of ESC and octal 335 (decimal 221) is sent instead. When the last byte in the packet has been sent, an END character is then transmitted.

Phil Karn suggests a simple change to the algorithm, which is to begin as well as end packets with an END character. This will flush any erroneous bytes which have been caused by line noise. In the normal case, the receiver will simply see two back-to-back END characters, which will generate a bad IP packet. If the SLIP implementation does not throw away the zero-length IP packet, the IP implementation certainly will. If there was line noise, the data received due to it will be discarded without affecting the following packet.

Because there is no 'standard' SLIP specification, there is no real defined maximum packet size for SLIP. It is probably best to accept the maximum packet size used by the Berkeley UNIX SLIP drivers: 1006 bytes including the IP and transport protocol headers (not including the framing characters). Therefore any new SLIP implementations should be prepared to accept 1006 byte datagrams and should not send more than 1006 bytes in a datagram.

RFC-1055 Transmission of IP Datagrams over Serial Lines (SLIP)

Deficiencies

There are several features that many users would like SLIP to provide which it doesn't. In all fairness, SLIP is just a very simple protocol designed quite a long time ago when these problems were not really important issues. The following are commonly perceived shortcomings in the existing SLIP protocol:

addressing:
both computers in a SLIP link need to know each other's IP addresses for routing purposes. Also, when using SLIP for hosts to dial-up a router, the addressing scheme may be quite dynamic and the router may need to inform the dialing host of the host's IP address. SLIP currently provides no mechanism for hosts to communicate addressing information over a SLIP connection.

type identification:
SLIP has no type field. Thus, only one protocol can be run over a SLIP connection, so in a configuration of two DEC computers running both TCP/IP and DECnet, there is no hope of having TCP/IP and DECnet share one serial line between them while using SLIP. While SLIP is "Serial Line IP", if a serial line connects two multi-protocol computers, those computers should be able to use more than one protocol over the line.

error detection/correction:
noisy phone lines will corrupt packets in transit. Because the line speed is probably quite low (likely 2400 baud), retransmitting a packet is very expensive. Error detection is not absolutely necessary at the SLIP level because any IP application should detect damaged packets (IP header and UDP and TCP checksums should suffice), although some common applications like NFS usually ignore the checksum and depend on the network media to detect damaged packets. Because it takes so long to retransmit a packet which was corrupted by line noise, it would be efficient if SLIP could provide some sort of simple error correction mechanism of its own.

compression:
because dial-in lines are so slow (usually 2400bps), packet compression would cause large improvements in packet throughput. Usually, streams of packets in a single TCP connection have few changed fields in the IP and TCP headers, so a simple compression algorithms might just send the changed parts of the headers instead of the complete headers.

Some work is being done by various groups to design and implement a successor to SLIP which will address some or all of these problems.

RFC-1055 Transmission of IP Datagrams over Serial Lines (SLIP)

Slip Drivers

The following C language functions send and receive SLIP packets. They depend on two functions, send_char() and recv_char(), which send and receive a single character over the serial line.

```
/* SLIP special character codes
 */
#define END             0300    /* indicates end of packet */
#define ESC             0333    /* indicates byte stuffing */
#define ESC_END         0334    /* ESC ESC_END means END data byte */
#define ESC_ESC         0335    /* ESC ESC_ESC means ESC data byte */

/* SEND_PACKET: sends a packet of length "len", starting at
 * location "p".
 */
void send_packet(p, len)
      char *p;
      int len;
   {

   /* send an initial END character to flush out any data that may
    * have accumulated in the receiver due to line noise
    */
      send_char(END);

   /* for each byte in the packet, send the appropriate character
    * sequence
    */
      while(len--) {
         switch(*p) {

         /* if it's the same code as an END character, we send a
          * special two character code so as not to make the
          * receiver think we sent an END
          */
         case END:
            send_char(ESC);
            send_char(ESC_END);
            break;

         /* if it's the same code as an ESC character,
          * we send a special two character code so as not
          * to make the receiver think we sent an ESC
          */
         case ESC:
            send_char(ESC);
            send_char(ESC_ESC);
            break;
         /* otherwise, we just send the character
          */
         default:
            send_char(*p);
```

```
         }
         p++;
      }

      /* tell the receiver that we're done sending the packet
       */
      send_char(END);
   }

   /* RECV_PACKET: receives a packet into the buffer located at "p".
    *      If more than len bytes are received, the packet will
    *      be truncated.
    *      Returns the number of bytes stored in the buffer.
    */
   int recv_packet(p, len)
      char *p;
      int len; {
      char c;
      int received = 0;

      /* sit in a loop reading bytes until we put together
       * a whole packet.
       * Make sure not to copy them into the packet if we
       * run out of room.
       */
      while(1) {
         /* get a character to process
          */
         c = recv_char();

         /* handle bytestuffing if necessary
          */
         switch(c) {

/* if it's an END character then we're done with
 * the packet
 */
            case END:
               /* a minor optimization: if there is no
                * data in the packet, ignore it. This is
                * meant to avoid bothering IP with all
                * the empty packets generated by the
                * duplicate END characters which are in
                * turn sent to try to detect line noise.
                */
               if(received)
                       return received;
               else
                       break;

            /* if it's the same code as an ESC character, wait
             * and get another character and then figure out
             * what to store in the packet based on that.
             */
            case ESC:
               c = recv_char();
```

```
            /* if "c" is not one of these two, then we
             * have a protocol violation.  The best bet
             * seems to be to leave the byte alone and
             * just stuff it into the packet
             */
            switch(c) {
            case ESC_END:
               c = END;
               break;
            case ESC_ESC:
               c = ESC;
               break;
            }

         /* here we fall into the default handler and let
          * it store the character for us
          */
         default:
            if(received < len)
               p[received++] = c;
      }
   }
}
```

RFC-1057
RPC: Remote Procedure Call Protocol Specification Version 2

Network Working Group
Sun Microsystems, Inc., June 1988
(Obsoletes: RFC 1050)

This RFC describes a standard that Sun Microsystems and others are using, and is one we wish to propose for the Internet's consideration. This memo is not an Internet standard at this time. Distribution of this memo is unlimited.

RFC-1057 Remote Procedure Call Version 2

Introduction

This document specifies version two of the message protocol used in Sun's Remote Procedure Call (RPC) package. The message protocol is specified with the eXternal Data Representation (XDR) language. This document assumes that the reader is familiar with XDR. It does not attempt to justify remote procedure calls systems or describe their use. The paper by Birrell and Nelson is recommended as an excellent background for the remote procedure call concept.

Terminology

This document discusses clients, calls, servers, replies, services, programs, procedures, and versions. Each remote procedure call has two sides: an active client side that sends the call to a server, which sends back a reply. A network service is a collection of one or more remote programs. A remote program implements one or more remote procedures; the procedures, their parameters, and results are documented in the specific program's protocol specification (see Appendix for an example). A server may support more than one version of a remote program in order to be compatible with changing protocols.

For example, a network file service may be composed of two programs. One program may deal with high-level applications such as file system access control and locking. The other may deal with low-level file input and output and have procedures like "read" and "write". A client of the network file service would call the procedures associated with the two programs of the service on behalf of the client.

The terms client and server only apply to a particular transaction; a particular hardware entity (host) or software entity (process or program) could operate in both roles at different times. For example, a program that supplies remote execution service could also be a client of a network file service. On the other hand, it may simplify software to separate client and server functionality into separate libraries or programs.

RFC-1057 Remote Procedure Call Version 2

The RPC Model

The Sun RPC protocol is based on the remote procedure call model, which is similar to the local procedure call model. In the local case, the caller places arguments to a procedure in some well- specified location (such as a register window). It then transfers control to the procedure, and eventually regains control. At that point, the results of the procedure are extracted from the well- specified location, and the caller continues execution.

The remote procedure call model is similar. One thread of control logically winds through two processes: the caller's process, and a server's process. The caller process first sends a call message to the server process and waits (blocks) for a reply message. The call message includes the procedure's parameters, and the reply message includes the procedure's results. Once the reply message is received, the results of the procedure are extracted, and caller's execution is resumed.

On the server side, a process is dormant awaiting the arrival of a call message. When one arrives, the server process extracts the procedure's parameters, computes the results, sends a reply message, and then awaits the next call message.

In this model, only one of the two processes is active at any given time. However, this model is only given as an example. The Sun RPC protocol makes no restrictions on the concurrency model implemented, and others are possible. For example, an implementation may choose to have RPC calls be asynchronous, so that the client may do useful work while waiting for the reply from the server. Another possibility is to have the server create a separate task to process an incoming call, so that the original server can be free to receive other requests.

There are a few important ways in which remote procedure calls differ from local procedure calls:

1. Error handling: failures of the remote server or network must be handled when using remote procedure calls.

2. Global variables and side-effects: since the server does not have access to the client's address space, hidden arguments cannot be passed as global variables or returned as side effects.

3. Performance: remote procedures usually operate one or more orders of magnitude slower than local procedure calls.

4. Authentication: since remote procedure calls can be transported over insecure networks, authentication may be necessary.

The conclusion is that even though there are tools to automatically generate client and server libraries for a given service, protocols must still be designed carefully.

Transports and Semantics

The RPC protocol can be implemented on several different transport protocols. The RPC protocol does not care how a message is passed from one process to another, but only with specification and interpretation of messages. On the other hand, the application may wish to obtain information about (and perhaps control over) the transport layer through an interface not specified in this document. For example, the transport protocol may impose a restriction on the maximum size of RPC messages, or it may be stream-oriented like TCP with no size limit. The client and server must agree on their transport protocol choices, through a mechanism such as the one described in Appendix A.

It is important to point out that RPC does not try to implement any kind of reliability and that the application may need to be aware of the type of transport protocol underneath RPC. If it knows it is running on top of a reliable transport such as TCP, then most of the work is already done for it. On the other hand, if it is running on top of an unreliable transport such as UDP, it must implement its own time-out, retransmission, and duplicate detection policies as the RPC layer does not provide these services.

Because of transport independence, the RPC protocol does not attach specific semantics to the remote procedures or their execution requirements. Semantics can be inferred from (but should be explicitly specified by) the underlying transport protocol. For example, consider RPC running on top of an unreliable transport such as UDP. If an application retransmits RPC call messages after time- outs, and does not receive a reply, it cannot infer anything about the number of times the procedure was executed. If it does receive a reply, then it can infer that the procedure was executed at least once.

A server may wish to remember previously granted requests from a client and not regrant them in order to insure some degree of execute-at-most-once semantics. A server can do this by taking advantage of the transaction ID that is packaged with every RPC message. The main use of this transaction is by the client RPC layer in matching replies to calls. However, a client application may choose to reuse its previous transaction ID when retransmitting a call. The server may choose to remember this ID after executing a call and not execute calls with the same ID in order to achieve some degree of execute-at-most-once semantics. The server is not allowed to examine this ID in any other way except as a test for equality.

On the other hand, if using a "reliable" transport such as TCP, the application can infer from a reply message that the procedure was executed exactly once, but if it receives no reply message, it cannot assume the remote procedure was not executed. Note that even if a connection-oriented protocol like TCP is used, an application still needs time-outs and reconnection to handle server crashes.

There are other possibilities for transports besides datagram- or connection-oriented protocols. For example, a request-reply protocol such as VMTP is perhaps a natural transport for RPC. The Sun RPC package currently uses both TCP and UDP transport protocols, with experimentation underway on others such as ISO TP4 and TP0.

Binding and Rendezvous Independence

The act of binding a particular client to a particular service and transport parameters is NOT part of this RPC protocol specification. This important and necessary function is left up to some higher-level software. (The software may use RPC itself; see Appendix.)

Implementors could think of the RPC protocol as the jump-subroutine instruction ("JSR") of a network; the loader (binder) makes JSR useful, and the loader itself uses JSR to accomplish its task. Likewise, the binding software makes RPC useful, possibly using RPC to accomplish this task.

Authentication

The RPC protocol provides the fields necessary for a client to identify itself to a service, and vice-versa, in each call and reply message. Security and access control mechanisms can be built on top of this message authentication. Several different authentication protocols can be supported. A field in the RPC header indicates which protocol is being used. More information on specific authentication protocols is in the section "Authentication Protocols".

RFC-1057 Remote Procedure Call Version 2

RPC Protocols

RPC Protocol Requirements

The RPC protocol must provide for the following:

1. Unique specification of a procedure to be called.
2. Provisions for matching response messages to request messages.
3. Provisions for authenticating the caller to service and vice- versa.

Besides these requirements, features that detect the following are worth supporting because of protocol roll-over errors, implementation bugs, user error, and network administration:

1. RPC protocol mismatches.
2. Remote program protocol version mismatches.
3. Protocol errors (such as misspecification of a procedure's parameters).
4. Reasons why remote authentication failed.
5. Any other reasons why the desired procedure was not called.

RFC-1057 Remote Procedure Call Version 2

RPC Programs and Procedures

The RPC call message has three unsigned integer fields -- remote program number, remote program version number, and remote procedure number -- which uniquely identify the procedure to be called. Program numbers are administered by some central authority (like Sun). Once implementors have a program number, they can implement their remote program; the first implementation would most likely have the version number 1. Because most new protocols evolve, a version field of the call message identifies which version of the protocol the caller is using. Version numbers make speaking old and new protocols through the same server process possible.

The procedure number identifies the procedure to be called. These numbers are documented in the specific program's protocol specification. For example, a file service's protocol specification may state that its procedure number 5 is "read" and procedure number 12 is "write".

Just as remote program protocols may change over several versions, the actual RPC message protocol could also change. Therefore, the call message also has in it the RPC version number, which is always equal to two for the version of RPC described here.

The reply message to a request message has enough information to distinguish the following error conditions:

1. The remote implementation of RPC does not speak protocol version 2. The lowest and highest supported RPC version numbers are returned.

2. The remote program is not available on the remote system.

3. The remote program does not support the requested version number. The lowest and highest supported remote program version numbers are returned.

4. The requested procedure number does not exist. (This is usually a client side protocol or programming error.)

5. The parameters to the remote procedure appear to be garbage from the server's point of view. (Again, this is usually caused by a disagreement about the protocol between client and service.)

Authentication

Provisions for authentication of caller to service and vice-versa are provided as a part of the RPC protocol. The call message has two authentication fields, the credentials and verifier. The reply message has one authentication field, the response verifier. The RPC protocol specification defines all three fields to be the following opaque type (in the eXternal Data Representation (XDR) language:

```
enum auth_flavor {
   AUTH_NULL       = 0,
   AUTH_UNIX       = 1,
   AUTH_SHORT      = 2,
   AUTH_DES        = 3
   /* and more to be defined */
};

struct opaque_auth {
   auth_flavor flavor;
   opaque body<400>;
};
```

In other words, any "opaque_auth" structure is an "auth_flavor" enumeration followed by bytes which are opaque to (uninterpreted by) the RPC protocol implementation.

The interpretation and semantics of the data contained within the authentication fields is specified by individual, independent authentication protocol specifications. (See the section on "Authentication Protocols".)

If authentication parameters were rejected, the reply message contains information stating why they were rejected.

Program Number Assignment

Program numbers are given out in groups of hexadecimal 20000000 (decimal 536870912) according to the following chart:

```
          0 - 1fffffff   defined by Sun
20000000 - 3fffffff   defined by user
40000000 - 5fffffff   transient
60000000 - 7fffffff   reserved
80000000 - 9fffffff   reserved
a0000000 - bfffffff   reserved
c0000000 - dfffffff   reserved
e0000000 - ffffffff   reserved
```

The first group is a range of numbers administered by Sun Microsystems and should be identical for all sites. The second range is for applications peculiar to a particular site. This range is intended primarily for debugging new programs. When a site develops an application that might be of general interest, that application should be given an assigned number in the first range. The third group is for applications that generate program numbers dynamically. The final groups are reserved for future use, and should not be used.

Other Uses of the RPC Protocol

The intended use of this protocol is for calling remote procedures. Normally, each call message is matched with a reply message. However, the protocol itself is a message-passing protocol with which other (non-procedure call) protocols can be implemented. Sun currently uses, or perhaps abuses, the RPC message protocol for the batching (or pipelining) and broadcast remote procedure calls.

Batching

Batching is useful when a client wishes to send an arbitrarily large sequence of call messages to a server. Batching typically uses reliable byte stream protocols (like TCP) for its transport. In the case of batching, the client never waits for a reply from the server, and the server does not send replies to batch calls. A sequence of batch calls is usually terminated by a legitimate remote procedure call operation in order to flush the pipeline and get positive acknowledgement.

Broadcast Remote Procedure Calls

In broadcast protocols, the client sends a broadcast call to the network and waits for numerous replies. This requires the use of packet-based protocols (like UDP) as its transport protocol. Servers that support broadcast protocols only respond when the call is successfully processed, and are silent in the face of errors. Broadcast calls use the Port Mapper RPC service to achieve their semantics. See Appendix for more information.

RFC-1057 Remote Procedure Call Version 2

The RPC Message Protocol

This section defines the RPC message protocol in the XDR data description language.

```
enum msg_type {
   CALL  = 0,
   REPLY = 1
};
```

A reply to a call message can take on two forms: The message was either accepted or rejected.

```
enum reply_stat {
   MSG_ACCEPTED = 0,
   MSG_DENIED   = 1
};
```

Given that a call message was accepted, the following is the status of an attempt to call a remote procedure.

```
enum accept_stat {
   SUCCESS       = 0, /* RPC executed successfully       */
   PROG_UNAVAIL  = 1, /* remote hasn't exported program  */
   PROG_MISMATCH = 2, /* remote can't support version #  */
   PROC_UNAVAIL  = 3, /* program can't support procedure */
   GARBAGE_ARGS  = 4  /* procedure can't decode params   */
};
```

Reasons why a call message was rejected:

```
enum reject_stat {
   RPC_MISMATCH = 0, /* RPC version number != 2          */
   AUTH_ERROR = 1    /* remote can't authenticate caller */
};
```

Why authentication failed:

```
enum auth_stat {
   AUTH_BADCRED      = 1,  /* bad credentials (seal broken) */
   AUTH_REJECTEDCRED = 2,  /* client must begin new session */
   AUTH_BADVERF      = 3,  /* bad verifier (seal broken)    */
   AUTH_REJECTEDVERF = 4,  /* verifier expired or replayed  */
   AUTH_TOOWEAK      = 5   /* rejected for security reasons */
};
```

The RPC message:

All messages start with a transaction identifier, `xid`, followed by a two-armed discriminated union. The union's discriminant is a `msg_type` which switches to one of the two types of the message. The `xid` of a `REPLY` message always matches that of the initiating `CALL` message. NB: The `xid` field is only used for clients matching reply messages with call messages or for servers detecting retransmissions; the service side cannot treat this id as any type of sequence number.

```
struct rpc_msg {
   unsigned int xid;
   union switch (msg_type mtype) {
   case CALL:
      call_body cbody;
   case REPLY:
      reply_body rbody;
   } body;
};
```

Body of an RPC call:

In version 2 of the RPC protocol specification, `rpcvers` must be equal to 2. The fields `prog`, `vers`, and `proc` specify the remote program, its version number, and the procedure within the remote program to be called. After these fields are two authentication parameters: `cred` (authentication credentials) and `verf` (authentication verifier). The two authentication parameters are followed by the parameters to the remote procedure, which are specified by the specific program protocol.

```
struct call_body {
   unsigned int rpcvers;       /* must be equal to two (2) */
   unsigned int prog;
   unsigned int vers;
   unsigned int proc;
   opaque_auth cred;
   opaque_auth verf;
   /* procedure specific parameters start here */
};
```

Body of a reply to an RPC call:

```
union reply_body switch (reply_stat stat) {
case MSG_ACCEPTED:
   accepted_reply areply;
case MSG_DENIED:
   rejected_reply rreply;
} reply;
```

Reply to an RPC call that was accepted by the server:

There could be an error even though the call was accepted. The first field is an authentication verifier that the server generates in order to validate itself to the client. It is followed by a union whose discriminant is an `enum accept_stat`. The `SUCCESS` arm of the union is protocol specific. The `PROG_UNAVAIL`, `PROC_UNAVAIL`, and `GARBAGE_ARGS` arms of the union are void. The `PROG_MISMATCH` arm specifies the lowest and highest version numbers of the remote program supported by the server.

```
struct accepted_reply {
   opaque_auth verf;
   union switch (accept_stat stat) {
   case SUCCESS:
      opaque results[0];
      /*
       * procedure-specific results start here
       */
    case PROG_MISMATCH:
```

```
            struct {
               unsigned int low;
               unsigned int high;
            } mismatch_info;
         default:
            /*
             * Void.  Cases include PROG_UNAVAIL, PROC_UNAVAIL,
             * and GARBAGE_ARGS.
             */
            void;
         } reply_data;
};
```

Reply to an RPC call that was rejected by the server:

The call can be rejected for two reasons: either the server is not running a compatible version of the RPC protocol (`RPC_MISMATCH`), or the server refuses to authenticate the caller (`AUTH_ERROR`). In case of an RPC version mismatch, the server returns the lowest and highest supported RPC version numbers. In case of refused authentication, failure status is returned.

```
union rejected_reply switch (reject_stat stat) {
case RPC_MISMATCH:
   struct {
      unsigned int low;
      unsigned int high;
   } mismatch_info;
case AUTH_ERROR:
   auth_stat stat;
};
```

Authentication Protocols

As previously stated, authentication parameters are opaque, but open-ended to the rest of the RPC protocol. This section defines some "flavors" of authentication implemented at (and supported by) Sun. Other sites are free to invent new authentication types, with the same rules of flavor number assignment as there is for program number assignment.

Null Authentication

Often calls must be made where the client does not know its identity or the server does not care who the client is. In this case, the flavor value (the discriminant of the opaque_auth's union) of the RPC message's credentials, verifier, and reply verifier is "AUTH_NULL". The bytes of the opaque_auth's body are undefined. It is recommended that the opaque length be zero.

UNIX Authentication

The client may wish to identify itself as it is identified on a UNIX(tm) system. The value of the credential's discriminant of an RPC call message is "AUTH_UNIX". The bytes of the credential's opaque body encode the the following structure:

```
struct auth_unix {
   unsigned int stamp;
   string machinename<255>;
   unsigned int uid;
   unsigned int gid;
   unsigned int gids<16>;
};
```

The "stamp" is an arbitrary ID which the caller machine may generate. The "machinename" is the name of the caller's machine (like "krypton"). The "uid" is the caller's effective user ID. The "gid" is the caller's effective group ID. The "gids" is a counted array of groups which contain the caller as a member. The verifier accompanying the credentials should be of "AUTH_NULL" (defined above). Note these credentials are only unique within a particular domain of machine names, uids, and gids. Inter-domain naming is beyond the scope of this document.

The value of the discriminant of the reply verifier received in the reply message from the server may be "AUTH_NULL" or "AUTH_SHORT". In the case of "AUTH_SHORT", the bytes of the reply verifier's string encode an opaque structure. This new opaque structure may now be passed to the server instead of the original "AUTH_UNIX" flavor credentials. The server may keep a cache which maps shorthand opaque structures (passed back by way of an "AUTH_SHORT" style reply verifier) to the original credentials of the caller. The caller can save network bandwidth and server cpu cycles by using the new credentials.

The server may flush the shorthand opaque structure at any time. If this happens, the remote procedure call message will be rejected due to an authentication error. The reason for the failure will be "AUTH_REJECTEDCRED". At this point, the client may wish to try the original "AUTH_UNIX" style of credentials.

RFC-1057 Remote Procedure Call Version 2: Authentication Protocols

DES Authentication

UNIX authentication suffers from three major problems:

1. The naming is too UNIX oriented.
2. There is no universal name, uid, and gid space.
3. There is no verifier, so credentials can easily be faked.

DES authentication attempts to address these problems.

Naming

The first problem is handled by addressing the client by a simple string of characters instead of by an operating system specific integer. This string of characters is known as the "`netname`" or network name of the client. The server is not allowed to interpret the contents of the client's name in any other way except to identify the client. Thus, netnames should be unique for every client in the Internet.

It is up to each operating system's implementation of DES authentication to generate netnames for its users that insure this uniqueness when they call upon remote servers. Operating systems already know how to distinguish users local to their systems. It is usually a simple matter to extend this mechanism to the network. For example, a UNIX user at Sun with a user ID of 515 might be assigned the following netname: "unix.515@sun.com". This netname contains three items that serve to insure it is unique. Going backwards, there is only one naming domain called "`sun.com`" in the Internet. Within this domain, there is only one UNIX user with user ID 515. However, there may be another user on another operating system, for example VMS, within the same naming domain that, by coincidence, happens to have the same user ID. To insure that these two users can be distinguished we add the operating system name. So one user is "`unix.515@sun.com`" and the other is "`vms.515@sun.com`".

The first field is actually a naming method rather than an operating system name. It happens that today there is almost a one-to-one correspondence between naming methods and operating systems. If the world could agree on a naming standard, the first field could be the name of that standard, instead of an operating system name.

RFC-1057 Remote Procedure Call Version 2: Authentication Protocols

DES Authentication Verifiers

Unlike UNIX authentication, DES authentication does have a verifier so the server can validate the client's credential (and vice-versa). The contents of this verifier is primarily an encrypted timestamp. The server can decrypt this timestamp, and if it is close to the real time, then the client must have encrypted it correctly. The only way the client could encrypt it correctly is to know the "conversation key" of the RPC session. And if the client knows the conversation key, then it must be the real client.

The conversation key is a DES key which the client generates and passes to the server in its first RPC call. The conversation key is encrypted using a public key scheme in this first transaction. The particular public key scheme used in DES authentication is Diffie- Hellman with 192-bit keys. The details of this encryption method are described later.

The client and the server need the same notion of the current time in order for all of this to work, perhaps by using the Network Time Protocol. If network time synchronization cannot be guaranteed, then the client can determine the server's time before beginning the conversation using a simpler time request protocol.

The way a server determines if a client timestamp is valid is somewhat complicated. For any other transaction but the first, the server just checks for two things:

1. the timestamp is greater than the one previously seen from the same client.

2. the timestamp has not expired.

A timestamp is expired if the server's time is later than the sum of the client's timestamp plus what is known as the client's "window". The "window" is a number the client passes (encrypted) to the server in its first transaction. You can think of it as a lifetime for the credential.

This explains everything but the first transaction. In the first transaction, the server checks only that the timestamp has not expired. If this was all that was done though, then it would be quite easy for the client to send random data in place of the timestamp with a fairly good chance of succeeding. As an added check, the client sends an encrypted item in the first transaction known as the "window verifier" which must be equal to the window minus 1, or the server will reject the credential.

The client too must check the verifier returned from the server to be sure it is legitimate. The server sends back to the client the encrypted timestamp it received from the client, minus one second. If the client gets anything different than this, it will reject it.

Nicknames and Clock Synchronization

After the first transaction, the server's DES authentication subsystem returns in its verifier to the client an integer "`nickname`" which the client may use in its further transactions instead of passing its netname, encrypted DES key and window every time. The nickname is most likely an index into a table on the server which stores for each client its netname, decrypted DES key and window.

Though they originally were synchronized, the client's and server's clocks can get out of sync again. When this happens the client RPC subsystem most likely will get back "`RPC_AUTHERROR`" at which point it should resynchronize.

A client may still get the "`RPC_AUTHERROR`" error even though it is synchronized with the server. The reason is that the server's nickname table is a limited size, and it may flush entries whenever it wants. A client should resend its original credential in this case and the server will give it a new nickname. If a server crashes, the entire nickname table gets flushed, and all clients will have to resend their original credentials.

DES Authentication Protocol Specification

There are two kinds of credentials: one in which the client uses its full network name, and one in which it uses its "nickname" (just an unsigned integer) given to it by the server. The client must use its fullname in its first transaction with the server, in which the server will return to the client its nickname. The client may use its nickname in all further transactions with the server. There is no requirement to use the nickname, but it is wise to use it for performance reasons.

```
enum authdes_namekind {
   ADN_FULLNAME = 0,
   ADN_NICKNAME = 1
};
```

A 64-bit block of encrypted DES data:

```
typedef opaque des_block[8];
```

Maximum length of a network user's name:

```
const MAXNETNAMELEN = 255;
```

A `fullname` contains the network name of the client, an encrypted conversation key and the window. The window is actually a lifetime for the credential. If the time indicated in the verifier timestamp plus the window has past, then the server should expire the request and not grant it. To insure that requests are not replayed, the server should insist that timestamps are greater than the previous one seen, unless it is the first transaction. In the first transaction, the server checks instead that the window verifier is one less than the window.

```
struct authdes_fullname {
   string name<MAXNETNAMELEN>;  /* name of client                */
   des_block key;               /* PK encrypted conversation key */
   opaque window[4];            /* encrypted window              */
};
```

A credential is either a fullname or a nickname:

```
union authdes_cred switch (authdes_namekind adc_namekind) {
case ADN_FULLNAME:
   authdes_fullname adc_fullname;
case ADN_NICKNAME:
   int adc_nickname;
};
```

A timestamp encodes the time since midnight, March 1, 1970.

```
struct timestamp {
   unsigned int seconds;    /* seconds          */
   unsigned int useconds;   /* and microseconds */
};
```

Verifier: client variety.

The window verifier is only used in the first transaction. In conjunction with a fullname credential, these items are packed into the following structure before being encrypted:

```
struct {
    adv_timestamp;        -- one DES block
    adc_fullname.window;  -- one half DES block
    adv_winverf;          -- one half DES block
}
```

This structure is encrypted using CBC mode encryption with an input vector of zero. All other encryptions of timestamps use ECB mode encryption.

```
struct authdes_verf_clnt {
   des_block adv_timestamp;    /* encrypted timestamp       */
   opaque adv_winverf[4];      /* encrypted window verifier */
};
```

Verifier: server variety.

The server returns (encrypted) the same timestamp the client gave it minus one second. It also tells the client its nickname to be used in future transactions (unencrypted).

```
struct authdes_verf_svr {
   des_block adv_timeverf;     /* encrypted verifier      */
   int adv_nickname;      /* new nickname for client */
};
```

RFC-1057 Remote Procedure Call Version 2: Authentication Protocols

Diffie-Hellman Encryption

In this scheme, there are two constants "BASE" and "MODULUS". The particular values Sun has chosen for these for the DES authentication protocol are:

```
const BASE = 3;
const MODULUS = "d4a0ba0250b6fd2ec626e7efd637df76c716e22d0944b88b"
```

The way this scheme works is best explained by an example. Suppose there are two people "A" and "B" who want to send encrypted messages to each other. So, A and B both generate "secret" keys at random which they do not reveal to anyone. Let these keys be represented as SK(A) and SK(B). They also publish in a public directory their "public" keys. These keys are computed as follows:

```
PK(A) = ( BASE ** SK(A) ) mod MODULUS
PK(B) = ( BASE ** SK(B) ) mod MODULUS
```

The "**" notation is used here to represent exponentiation. Now, both A and B can arrive at the "common" key between them, represented here as CK(A, B), without revealing their secret keys.

A computes:

```
CK(A, B) = ( PK(B) ** SK(A)) mod MODULUS
```

while B computes:

```
CK(A, B) = ( PK(A) ** SK(B)) mod MODULUS
```

These two can be shown to be equivalent:

```
(PK(B) ** SK(A)) mod MODULUS = (PK(A) ** SK(B)) mod MODULUS
```

We drop the "mod MODULUS" parts and assume modulo arithmetic to simplify things:

```
PK(B) ** SK(A) = PK(A) ** SK(B)
```

Then, replace PK(B) by what B computed earlier and likewise for PK(A).

```
((BASE ** SK(B)) ** SK(A) = (BASE ** SK(A)) ** SK(B)
```

which leads to:

```
BASE ** (SK(A) * SK(B)) = BASE ** (SK(A) * SK(B))
```

This common key CK(A, B) is not used to encrypt the timestamps used in the protocol. Rather, it is used only to encrypt a conversation key which is then used to encrypt the timestamps. The reason for doing this is to use the common key as little as possible, for fear that it could be broken. Breaking the conversation key is a far less serious offense, since conversations are relatively short-lived.

The conversation key is encrypted using 56-bit DES keys, yet the common key is 192 bits.

To reduce the number of bits, 56 bits are selected from the common key as follows. The middle-most 8-bytes are selected from the common key, and then parity is added to the lower order bit of each byte, producing a 56-bit key with 8 bits of parity.

RFC-1057 Remote Procedure Call Version 2

Record Marking Standard

When RPC messages are passed on top of a byte stream transport protocol (like TCP), it is necessary to delimit one message from another in order to detect and possibly recover from protocol errors. This is called record marking (RM). Sun uses this RM/TCP/IP transport for passing RPC messages on TCP streams. One RPC message fits into one RM record.

A record is composed of one or more record fragments. A record fragment is a four-byte header followed by 0 to (2**31) - 1 bytes of fragment data. The bytes encode an unsigned binary number; as with XDR integers, the byte order is from highest to lowest. The number encodes two values -- a boolean which indicates whether the fragment is the last fragment of the record (bit value 1 implies the fragment is the last fragment) and a 31-bit unsigned binary value which is the length in bytes of the fragment's data. The boolean value is the highest-order bit of the header; the length is the 31 low-order bits. (Note that this record specification is NOT in XDR standard form!)

The RPC Language

Just as there was a need to describe the XDR data-types in a formal language, there is also need to describe the procedures that operate on these XDR data-types in a formal language as well. The RPC Language is an extension to the XDR language, with the addition of "program", "procedure", and "version" declarations. The following example is used to describe the essence of the language.

An Example Service Described in the RPC Language

Here is an example of the specification of a simple ping program.

```
program PING_PROG {
      /*
       * Latest and greatest version
       */
      version PING_VERS_PINGBACK {
         void
         PINGPROC_NULL(void) = 0;

         /*
          * Ping the client, return the round-trip time
          * (in microseconds). Returns -1 if the operation
          * timed out.
          */
         int
         PINGPROC_PINGBACK(void) = 1;
      } = 2;

      /*
       * Original version
       */
      version PING_VERS_ORIG {
         void
         PINGPROC_NULL(void) = 0;
      } = 1;
   } = 1;

   const PING_VERS = 2;      /* latest version */
```

The first version described is `PING_VERS_PINGBACK` with two procedures, `PINGPROC_NULL` and `PINGPROC_PINGBACK`. `PINGPROC_NULL` takes no arguments and returns no results, but it is useful for computing round-trip times from the client to the server and back again. By convention, procedure 0 of any RPC protocol should have the same semantics, and never require any kind of authentication. The second procedure is used for the client to have the server do a reverse ping operation back to the client, and it returns the amount of time (in microseconds) that the operation used. The next version, `PING_VERS_ORIG`, is the original version of the protocol and it does not contain `PINGPROC_PINGBACK` procedure. It is useful for compatibility with old client programs, and as this program matures it may be dropped from the protocol entirely.

The RPC Language Specification

The RPC language is identical to the XDR language defined in RFC 1014, except for the added definition of a "`program-def`" described below.

```
program-def:
   "program" identifier "{"
      version-def
      version-def *
   "}" "=" constant ";"

version-def:
   "version" identifier "{"
       procedure-def
       procedure-def *
   "}" "=" constant ";"

procedure-def:
   type-specifier identifier "(" type-specifier
     ("," type-specifier )* ")" "=" constant ";"
```

Syntax Notes

1. The following keywords are added and cannot be used as identifiers: "`program`" and "`version`";

2. A version name cannot occur more than once within the scope of a program definition. Nor can a version number occur more than once within the scope of a program definition.

3. A procedure name cannot occur more than once within the scope of a version definition. Nor can a procedure number occur more than once within the scope of version definition.

4. Program identifiers are in the same name space as constant and type identifiers.

5. Only unsigned constants can be assigned to programs, versions and procedures.

Appendix: Port Mapper Program Protocol

The port mapper program maps RPC program and version numbers to transport-specific port numbers. This program makes dynamic binding of remote programs possible.

This is desirable because the range of reserved port numbers is very small and the number of potential remote programs is very large. By running only the port mapper on a reserved port, the port numbers of other remote programs can be ascertained by querying the port mapper.

The port mapper also aids in broadcast RPC. A given RPC program will usually have different port number bindings on different machines, so there is no way to directly broadcast to all of these programs. The port mapper, however, does have a fixed port number. So, to broadcast to a given program, the client actually sends its message to the port mapper located at the broadcast address. Each port mapper that picks up the broadcast then calls the local service specified by the client. When the port mapper gets the reply from the local service, it sends the reply on back to the client.

RFC-1057 Remote Procedure Call Version 2: Appendix

Port Mapper Protocol Specification (in RPC Language)

```
const PMAP_PORT = 111;      /* portmapper port number */
```

A mapping of (program, version, protocol) to port number:

```
struct mapping {
   unsigned int prog;
   unsigned int vers;
   unsigned int prot;
   unsigned int port;
};
```

Supported values for the "`prot`" field:

```
const IPPROTO_TCP = 6;      /* protocol number for TCP/IP */
const IPPROTO_UDP = 17;     /* protocol number for UDP/IP */
```

A list of mappings:

```
struct *pmaplist {
   mapping map;
   pmaplist next;
};
```

Arguments to callit:

```
struct call_args {
   unsigned int prog;
   unsigned int vers;
   unsigned int proc;
   opaque args<>;
};
```

Results of callit:

```
struct call_result {
   unsigned int port;
   opaque res<>;
};
```

Port mapper procedures:

```
program PMAP_PROG {
   version PMAP_VERS {
      void
      PMAPPROC_NULL(void)         = 0;

      bool
      PMAPPROC_SET(mapping)       = 1;

      bool
      PMAPPROC_UNSET(mapping)     = 2;
```

```
      unsigned int
      PMAPPROC_GETPORT(mapping)    = 3;

      pmaplist
      PMAPPROC_DUMP(void)          = 4;

      call_result
      PMAPPROC_CALLIT(call_args)   = 5;
   } = 2;
} = 100000;
```

RFC-1057 Remote Procedure Call Version 2: Appendix

Port Mapper Operation

The portmapper program currently supports two protocols (UDP and TCP). The portmapper is contacted by talking to it on assigned port number 111 (SUNRPC) on either of these protocols.

The following is a description of each of the portmapper procedures:

`PMAPPROC_NULL:`

This procedure does no work. By convention, procedure zero of any protocol takes no parameters and returns no results.

`PMAPPROC_SET:`

When a program first becomes available on a machine, it registers itself with the port mapper program on the same machine. The program passes its program number "`prog`", version number "`vers`", transport protocol number "`prot`", and the port "`port`" on which it awaits service request. The procedure returns a boolean reply whose value is "TRUE" if the procedure successfully established the mapping and "FALSE" otherwise. The procedure refuses to establish a mapping if one already exists for the tuple "`(prog, vers, prot)`".

`PMAPPROC_UNSET:`

When a program becomes unavailable, it should unregister itself with the port mapper program on the same machine. The parameters and results have meanings identical to those of "`PMAPPROC_SET`". The protocol and port number fields of the argument are ignored.

`PMAPPROC_GETPORT:`

Given a program number "`prog`", version number "`vers`", and transport protocol number "`prot`", this procedure returns the port number on which the program is awaiting call requests. A port value of zeros means the program has not been registered. The "`port`" field of the argument is ignored.

`PMAPPROC_DUMP:`

This procedure enumerates all entries in the port mapper's database. The procedure takes no parameters and returns a list of program, version, protocol, and port values.

`PMAPPROC_CALLIT:`

This procedure allows a client to call another remote procedure on the same machine without knowing the remote procedure's port number. It is intended for supporting broadcasts to arbitrary remote programs via the well-known port mapper's port. The parameters "`prog`", "`vers`", "`proc`", and the bytes of "`args`" are the program number, version number, procedure number, and parameters of the remote procedure.

Note:

1. This procedure only sends a reply if the procedure was successfully executed and is silent (no reply) otherwise.

2 The port mapper communicates with the remote program using UDP only.

The procedure returns the remote program's port number, and the reply is the reply of the remote procedure.

Birrell, A. D. & Nelson, B. J., "Implementing Remote Procedure Calls", XEROX CSL-83-7, October 1983.

Cheriton, D., "VMTP: Versatile Message Transaction Protocol", Preliminary Version 0.3, Stanford University, January 1987.

Diffie & Hellman, "New Directions in Cryptography", IEEE Transactions on Information Theory IT-22, November 1976.

National Bureau of Standards, "Data Encryption Standard", Federal Information Processing Standards Publication 46, January 1977.

RFC-1060 Assigned Numbers

March 1990

Status Of This Memo

This memo is a status report on the parameters (i.e., numbers and keywords) used in protocols in the Internet community. Distribution of this memo is unlimited.

Introduction

This Network Working Group Request for Comments documents the currently assigned values from several series of numbers used in network protocol implementations. This RFC will be updated periodically, and in any case current information can be obtained from the Internet Assigned Numbers Authority (IANA). If you are developing a protocol or application that will require the use of a link, socket, port, protocol, etc., please contact the IANA to receive a number assignment.

Joyce K. Reynolds
Internet Assigned Numbers Authority
USC - Information Sciences Institute
4676 Admiralty Way
Marina del Rey, California 90292-6695

Phone: (213) 822-1511
Electronic mail: JKREY@ISI.EDU

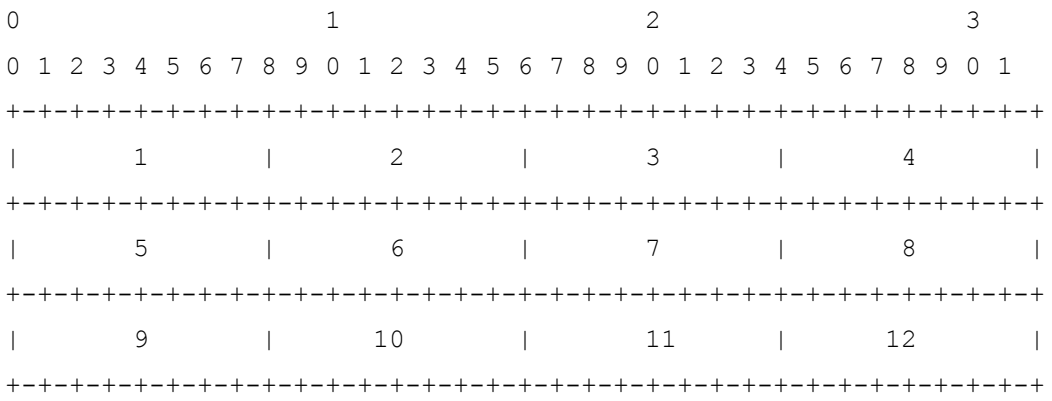
Most of the protocols mentioned here are documented in the RFC series of notes. Some of the items listed are undocumented. Further information on protocols can be found in the memo "Official Internet Protocols" [118]. The more prominent and more generally used are documented in the "DDN Protocol Handbook, Volume Two, DARPA Internet Protocols" [45] prepared by the NIC. Other collections of older or obsolete protocols are contained in the "Internet Protocol Transition Workbook" [76], or in the "ARPANET Protocol Transition Handbook" [47]. For further information on ordering the complete 1985 DDN Protocol Handbook, write: SRI International (SRI-NIC), DDN Network Information Center, Room EJ291, 333 Ravenswood Avenue, Menlo Park, CA., 94025; or call: 1-800-235-3155. Also, the Internet Activities Board (IAB) publishes the "IAB Official Protocol Standards" [62], which describes the state of standardization of protocols used in the Internet. This document is issued quarterly. Current copies may be obtained from the DDN Network Information Center or from the IANA.

In the entries below, the name and mailbox of the responsible individual is indicated. The bracketed entry, e.g., [nn,iii], at the right hand margin of the page indicates a reference for the listed protocol, where the number ("nn") cites the document and the letters ("iii") cites the person. Whenever possible, the letters are a NIC Ident as used in the WhoIs (NICNAME) service.

Data Notations

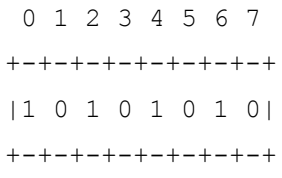
The convention in the documentation of Internet Protocols is to express numbers in decimal and to picture data in "big-endian" order [21]. That is, fields are described left to right, with the most significant octet on the left and the least significant octet on the right.

The order of transmission of the header and data described in this document is resolved to the octet level. Whenever a diagram shows a group of octets, the order of transmission of those octets is the normal order in which they are read in English. For example, in the following diagram the octets are transmitted in the order they are numbered.



Transmission Order of Bytes

Whenever an octet represents a numeric quantity the left most bit in the diagram is the high order or most significant bit. That is, the bit labeled 0 is the most significant bit. For example, the following diagram represents the value 170 (decimal).



Significance of Bits

Similarly, whenever a multi-octet field represents a numeric quantity the left most bit of the whole field is the most significant bit. When a multi-octet quantity is transmitted the most significant octet is transmitted first.

Special Addresses

There are five classes of IP addresses: Class A through Class E [119]. Of these, Class D and Class E addresses are reserved for experimental use. A gateway which is not participating in these experiments must ignore all datagrams with a Class D or Class E destination IP address. ICMP Destination Unreachable or ICMP Redirect messages must not result from receiving such datagrams.

There are certain special cases for IP addresses [11]. These special cases can be concisely summarized using the earlier notation for an IP address:

IP-address ::= { <Network-number>, <Host-number> }

or

IP-address ::= { <Network-number>, <Subnet-number>, <Host-number> }

if we also use the notation "-1" to mean the field contains all 1 bits. Some common special cases are as follows:

(a) {0, 0}

This host on this network. Can only be used as a source address (see note later).

(b) {0, <Host-number>}

Specified host on this network. Can only be used as a source address.

(c) { -1, -1}

Limited broadcast. Can only be used as a destination address, and a datagram with this address must never be forwarded outside the (sub-)net of the source.

(d) {<Network-number>, -1}

Directed broadcast to specified network. Can only be used as a destination address.

(e) {<Network-number>, <Subnet-number>, -1}

Directed broadcast to specified subnet. Can only be used as a destination address.

(f) {<Network-number>, -1, -1}

Directed broadcast to all subnets of specified subnetted network. Can only be used as a destination address.

(g) {127, <any>}

Internal host loopback address. Should never appear outside a host.

Version Numbers

In the Internet Protocol (IP) [45,105] there is a field to identify the version of the internetwork general protocol. This field is 4 bits in size.

Assigned Internet Version Numbers

Decimal	Keyword	Version	References
0		Reserved	[JBP]
1-3		Unassigned	[JBP]
4	IP	Internet Protocol	[105,JBP]
5	ST	ST Datagram Mode	[49,JWF]
6-14		Unassigned	[JBP]
15		Reserved	[JBP]

Protocol Numbers

In the Internet Protocol (IP) [45,105] there is a field, called Protocol, to identify the the next level protocol. This is an 8 bit field.

Assigned Internet Protocol Numbers

Decimal	Keyword	Protocol	References
0		Reserved	[JBP]
1	ICMP	Internet Control Message	[97,JBP]
2	IGMP	Internet Group Management	[43,JBP]
3	GGP	Gateway-to-Gateway	[60,MB]
4		Unassigned	[JBP]
5	ST	Stream	[49,JWF]
6	TCP	Transmission Control	[106,JBP]
7	UCL	UCL	[PK]
8	EGP	Exterior Gateway Protocol	[123,DLM1]
9	IGP	any private interior gateway	[JBP]
10	BBN-RCC-MON	BBN RCC Monitoring	[SGC]
11	NVP-II	Network Voice Protocol	[22,SC3]
12	PUP	PUP	[8,XEROX]
13	ARGUS	ARGUS	[RWS4]
14	EMCON	EMCON	[BN7]
15	XNET	Cross Net Debugger	[56,JFH2]
16	CHAOS	Chaos	[NC3]
17	UDP	User Datagram	[104,JBP]
18	MUX	Multiplexing	[23,JBP]
19	DCN-MEAS	DCN Measurement Subsystems	[DLM1]
20	HMP	Host Monitoring	[59,RH6]
21	PRM	Packet Radio Measurement	[ZSU]
22	XNS-IDP	XEROX NS IDP	[133,XEROX]
23	TRUNK-1	Trunk-1	[BWB6]
24	TRUNK-2	Trunk-2	[BWB6]
25	LEAF-1	Leaf-1	[BWB6]
26	LEAF-2	Leaf-2	[BWB6]
27	RDP	Reliable Data Protocol	[138,RH6]
28	IRTP	Internet Reliable Transaction	[79,TXM]
29	ISO-TP4	ISO Transport Protocol Class 4	[63,RC77]
30	NETBLT	Bulk Data Transfer Protocol	[20,DDC1]
31	MFE-NSP	MFE Network Services Protocol	[124,BCH2]
32	MERIT-INP	MERIT Internodal Protocol	[HWB]
33	SEP	Sequential Exchange Protocol	[JC120]
34	3PC	Third Party Connect Protocol	[SAF3]
35-60		Unassigned	[JBP]
61		any host internal protocol	[JBP]
62	CFTP	CFTP	[50,HCF2]
63		any local network	[JBP]
64	SAT-EXPAK	SATNET and Backroom EXPAK	[SHB]
65		Unassigned	[JBP]
66	RVD	MIT Remote Virtual Disk Protocol	[MBG]
67	IPPC	Internet Pluribus Packet Core	[SHB]
68		any distributed file system	[JBP]
69	SAT-MON	SATNET Monitoring	[SHB]

70	VISA	VISA Protocol	[GXT1]
71	IPCV	Internet Packet Core Utility	[SHB]
72-75		Unassigned	[JBP]
76	BR-SAT-MON	Backroom SATNET Monitoring	[SHB]
77	SUN-ND	SUN ND PROTOCOL-Temporary	[WM3]
78	WB-MON	WIDEBAND Monitoring	[SHB]
79	WB-EXPAK	WIDEBAND EXPAK	[SHB]
80	ISO-IP	ISO Internet Protocol	[MTR]
81	VMTP	VMTP	[DRC3]
82	SECURE-VMTP	SECURE-VMTP	[DRC3]
83	VINES	VINES	[BXH]
84	TTP	TTP	[JXS]
85	NSFNET-IGP	NSFNET-IGP	[HWB]
86	DGP	Dissimilar Gateway Protocol	[74,ML109]
87	TCF	TCF	[GAL5]
88	IGRP	IGRP	[18,GXS]
89	OSPFIGP	OSPFIGP	[83,JTM4]
90	Sprite-RPC	Sprite RPC Protocol	[143,BXW]
91	LARP	Locus Address Resolution	[BXH]
92-254		Unassigned	[JBP]
255		Reserved	[JBP]

Port Numbers

Ports are used in the TCP [45,106] to name the ends of logical connections which carry long term conversations. For the purpose of providing services to unknown callers, a service contact port is defined. This list specifies the port used by the server process as its contact port. The contact port is sometimes called the "well-known port". See also UNIX Ports.

To the extent possible, these same port assignments are used with the UDP [46,104].

To the extent possible, these same port assignments are used with the ISO-TP4 [64].

The assigned ports use a small portion of the possible port numbers.

The assigned ports have all except the low order eight bits cleared to zero. The low order eight bits are specified here.

Port Assignments:

Decimal	Keyword	Description	References
0		Reserved	[JBP]
1	TCPMUX	TCP Port Service Multiplexer	[MKL]
2-4		Unassigned	[JBP]
5	RJE	Remote Job Entry	[12,JBP]
7	ECHO	Echo	[95,JBP]
9	DISCARD	Discard	[94,JBP]
11	USERS	Active Users	[89,JBP]
13	DAYTIME	Daytime	[93,JBP]
15		Unassigned	[JBP]
17	QUOTE	Quote of the Day	[100,JBP]
19	CHARGEN	Character Generator	[92,JBP]
20	FTP-DATA	File Transfer [Default Data]	[96,JBP]
21	FTP	File Transfer [Control]	[96,JBP]
23	TELNET	Telnet	[112,JBP]
25	SMTP	Simple Mail Transfer	[102,JBP]
27	NSW-FE	NSW User System FE	[24,RHT]
29	MSG-ICP	MSG ICP	[85,RHT]
31	MSG-AUTH	MSG Authentication	[85,RHT]
33	DSP	Display Support Protocol	[EXC]
35		any private printer server	[JBP]
37	TIME	Time	[108,JBP]
39	RLP	Resource Location Protocol	[MA]
41	GRAPHICS	Graphics	[129,JBP]
42	NAMESERVER	Host Name Server	[99,JBP]
43	NICNAME	Who Is	[55,MARY]
44	MPM-FLAGS	MPM FLAGS Protocol	[JBP]
45	MPM	MPM [recv]	[98,JBP]
46	MPM-SND	MPM [default send]	[98,JBP]
47	NI-FTP	NI FTP	[134,SK8]
49	LOGIN	Login Host Protocol	[PHD1]
51	LA-MAINT	IMP Logical Address Maint	[76,AGM]
53	DOMAIN	Domain Name Server	[81,82,PM1]
55	ISI-GL	ISI Graphics Language	[7,RB9]
57		any private terminal access	[JBP]
59		any private file service	[JBP]
61	NI-MAIL	NI MAIL	[5,SK8]

63	VIA-FTP	VIA Systems - FTP	[DXD]
65	TACACS-DS	TACACS-Database Service	[3,KH43]
67	BOOTPS	Bootstrap Protocol Server	[36,WJC2]
68	BOOTPC	Bootstrap Protocol Client	[36,WJC2]
69	TFTP	Trivial File Transfer	[126,DDC1]
71	NETRJS-1	Remote Job Service	[10,RTB3]
72	NETRJS-2	Remote Job Service	[10,RTB3]
73	NETRJS-3	Remote Job Service	[10,RTB3]
74	NETRJS-4	Remote Job Service	[10,RTB3]
75		any private dial out service	[JBP]
77		any private RJE service	[JBP]
79	FINGER	Finger	[52,KLH]
81	HOSTS2-NS	HOSTS2 Name Server	[EAK1]
83	MIT-ML-DEV	MIT ML Device	[DPR]
85	MIT-ML-DEV	MIT ML Device	[DPR]
87		any private terminal link	[JBP]
89	SU-MIT-TG	SU/MIT Telnet Gateway	[MRC]
91	MIT-DOV	MIT Dover Spooler	[EBM]
93	DCP	Device Control Protocol	[DT15]
95	SUPDUP	SUPDUP	[27,MRC]
97	SWIFT-RVF	Swift Remote Vitural File	[MXR]
98	TACNEWS	TAC News	[ANM2]
99	METAGRAM	Metagram Relay	[GEOF]
101	HOSTNAME	NIC Host Name Server	[54,MARY]
102	ISO-TSAP	ISO-TSAP	[16,MTR]
103	X400	X400	[HCF2]
104	X400-SND	X400-SND	[HCF2]
105	CSNET-NS	Mailbox Name Nameserver	[127,MS56]
107	RTELNET	Remote Telnet Service	[101,JBP]
109	POP2	Post Office Protocol - Ver 2	[14,JKR1]
110	POP3	Post Office Protocol - Ver 3	[122,MTR]
111	SUNRPC	SUN Remote Procedure Call	[DXG]
113	AUTH	Authentication Service	[130,MCSJ]
115	SFTP	Simple File Transfer Protocol	[73,MKL1]
117	UUCP-PATH	UUCP Path Service	[44,MAE]
119	NNTP	Network News Transfer	[65,PL4]
121	ERPC	Encore Expedited RPC	[132,JXO]
123	NTP	Network Time Protocol	[80,DLM1]
125	LOCUS-MAP	Locus PC-Interface Net Map	[137,EP53]
127	LOCUS-CON	Locus PC-Interface Conn	[137,EP53]
129	PWDGEN	Password Generator Protocol	[141,FJW]
130	CISCO-FNA	CISCO FNATIVE	[WXB]
131	CISCO-TNA	CISCO TNATIVE	[WXB]
132	CISCO-SYS	CISCO SYSMAINT	[WXB]
133	STATSRV	Statistics Service	[DLM1]
134	INGRES-NET	INGRES-NET Service	[MXB]
135	LOC-SRV	Location Service	[JXP]
136	PROFILE	PROFILE Naming System	[LLP]
137	NETBIOS-NS	NETBIOS Name Service	[JBP]
138	NETBIOS-DGM	NETBIOS Datagram Service	[JBP]
139	NETBIOS-SSN	NETBIOS Session Service	[JBP]
140	EMFIS-DATA	EMFIS Data Service	[GB7]
141	EMFIS-CNTL	EMFIS Control Service	[GB7]
142	BL-IDM	Britton-Lee IDM	[SXS1]
143	IMAP2	InterimMail Access v2	[MRC]

144	NEWS	NewS	[JAG]
145	UAAC	UAAC Protocol	[DAG4]
146	ISO-TP0	ISO-IP0	[86,MTR]
147	ISO-IP	ISO-IP	[MTR]
148	CRONUS	CRONUS-SUPPORT	[135,JXB]
149	AED-512	AED 512 Emulation Service	[AXB]
150	SQL-NET	SQL-NET	[MXP]
151	HEMS	HEMS	[87,CXT]
152	BFTP	Background FTP	[AD14]
153	SGMP	SGMP	[37,MS9]
154	NETSC-PROD	NETSC	[SH37]
155	NETSC-DEV	NETSC	[SH37]
156	SQLSRV	SQL Service	[CMR]
157	KNET-CMP	KNET/VM Com/Mess	[77,GSM11]
158	PCMail-SRV	PCMail Server	[19,MXL]
159	NSS-Routing	NSS-Routing	[JXR]
160	SGMP-TRAPS	SGMP-TRAPS	[37,MS9]
161	SNMP	SNMP	[15,MTR]
162	SNMPTRAP	SNMPTRAP	[15,MTR]
163	CMIP-Manage	CMIP/TCP Manager	[4,AXB1]
164	CMIP-Agent	CMIP/TCP Agent	[4,AXB1]
165	XNS-Courier	Xerox	[144,SXA]
166	S-Net	Sirius Systems	[BXL]
167	NAMP	NAMP	[MS9]
168	RSVD	RSVD	[NT12]
169	SEND	SEND	[WDW11]
170	Print-SRV	Network PostScript	[BKR]
171	Multiplex	Network Innovations Multiplex	[KXD]
172	CL/1	Network Innovations CL/1	[KXD]
173	Xyplex-MUX	Xyplex	[BXS]
174	MAILQ	MAILQ	[RXZ]
175	VMNET	VMNET	[CXT]
176	GENRAD-MUX	GENRAD-MUX	[RXT]
177	XDMCP	X Display Manager Control	[RWS4]
178	NextStep	NextStep Window Server	[LXH]
179	BGP	Border Gateway Protocol	[KSL]
180	RIS	Intergraph	[DXB]
181	Unify	Unify	[VXS]
182	Unisys-Cam	Unisys-Cam	[GXG]
183	OCBinder	OCBinder	[JXO1]
184	OCServer	OCServer	[JXO1]
185	Remote-KIS	Remote-KIS	[RXD1]
186	KIS	KIS Protocol	[RXD1]
187	ACI	Application Comm Interface	[RXC1]
188	MUMPS	MUMPS	[HS23]
189	QFT	Queued File Transport	[WXS]
190	GACP	Gateway Access Control	[PCW]
191	Prospero	Prospero	[BCN]
192	OSU-NMS	OSU Network Monitoring	[DXK]
193	SRMP	Spider Remote Monitoring	[TXS]
194	IRC	Internet Relay Chat Protocol	[JXO2]
195	DN6-NLM-AUD	DNSIX Network Level Mo Audit	[LL69]
196	DN6-SMM-RED	DNSIX Sess Mgt Mod Audir Red	[LL69]
197	DLS	Directory Location Service	[SXB]
198	DLS-Mon	Directory Location Service Mon	[SXB]

198-200		Unassigned	[JBP]
201	AT-RMTP	AppleTalk Routing Maint	[RXC]
202	AT-NBP	AppleTalk Name Binding	[RXC]
203	AT-3	AppleTalk Unused	[RXC]
204	AT-ECHO	AppleTalk Echo	[RXC]
205	AT-5	AppleTalk Unused	[RXC]
206	AT-ZIS	AppleTalk Zone Information	[RXC]
207	AT-7	AppleTalk Unused	[RXC]
208	AT-8	AppleTalk Unused	[RXC]
209-223		Unassigned	[JBP]
224-241		Reserved	[JBP]
243	SUR-MEAS	Survey Measurement	[6,DDC1]
245	LINK	LINK	[1,RDB2]
246	DSP3270	Display Systems Protocol	[39,WJS1]
247-255		Reserved	[JBP]

UNIX Ports

By convention, ports in the range 256 to 1024 are used for "Unix Standard" services. Listed here are some of the normal uses of these port numbers.

Service Name	Port/Protocol	Description
TCP		
echo	7/tcp	
discard	9/tcp	sink null
systat	11/tcp	users
daytime	13/tcp	
netstat	15/tcp	
qotd	17/tcp	quote
chargen	19/tcp	ttytst source
ftp-data	20/tcp	
ftp	21/tcp	
telnet	23/tcp	
smtp	25/tcp	mail
time	37/tcp	timserver
name	42/tcp	nameserver
whois	43/tcp	nicname
nameserver	53/tcp	domain
apts	57/tcp	any private terminal service
apfs	59/tcp	any private file service
rje	77/tcp	netrjs
finger	79/tcp	
link	87/tcp	ttylink
supdup	95/tcp	
newacct	100/tcp	[unauthorized use]
hostnames	101/tcp	hostname
iso-tsap	102/tcp	tsap
x400	103/tcp	
x400-snd	104/tcp	
csnet-ns	105/tcp	CSNET Name Service
pop-2	109/tcp	pop postoffice
sunrpc	111/tcp	
auth	113/tcp	authentication
sftp	115/tcp	
uucp-path	117/tcp	
nntp	119/tcp	usenet readnews untp
ntp	123/tcp	network time protocol
statsrv	133/tcp	
profile	136/tcp	
NeWS	144/tcp	news
print-srv	170/tcp	
exec	512/tcp	remote process execution; authentication performed using passwords and UNIX login names
login	513/tcp	remotelogin a la telnet; automatic authentication performed based on priviledged port numbers and distributed data bases which identify "authentication domains"
cmd	514/tcp	like exec, but automatic authentication is performed as

		for login server
printer	515/tcp	spooler
efs	520/tcp	extended file name server
tempo	526/tcp	newdate
courier	530/tcp	rpc
conference	531/tcp	chat
netnews	532/tcp	readnews
uucp	540/tcp	uucpd
klogin	543/tcp	
kshell	544/tcp	krcmd
dsf	555/tcp	
remotefs	556/tcp	rfs server
chshell	562/tcp	chcmd
meter	570/tcp	demon
pcserver	600/tcp	Sun IPC server
nqs	607/tcp	nqs
mdqs	666/tcp	
rfile	750/tcp	
pump	751/tcp	
qrh	752/tcp	
rrh	753/tcp	
tell	754/tcp	send
nlogin	758/tcp	
con	759/tcp	
ns	760/tcp	
rxe	761/tcp	
quotad	762/tcp	
cycleserv	763/tcp	
omserv	764/tcp	
webster	765/tcp	
phonebook	767/tcp	phone
vid	769/tcp	
rtip	771/tcp	
cycleserv2	772/tcp	
submit	773/tcp	
rpasswd	774/tcp	
entomb	775/tcp	
wpages	776/tcp	
wpgs	780/tcp	
mdbs_daemon	800/tcp	
device	801/tcp	
maitrd	997/tcp	
busboy	998/tcp	
garcon	999/tcp	
blackjack	1025/tcp	network blackjack
bbn-mmc	1347/tcp	multi media conferencing
bbn-mmx	1348/tcp	multi media conferencing
orasrv	1525/tcp	oracle
ingreslock	1524/tcp	
issd	1600/tcp	
nkd	1650/tcp	
dc	2001/tcp	
mailbox	2004/tcp	
berknet	2005/tcp	
invokator	2006/tcp	

dectalk	2007/tcp	
conf	2008/tcp	
news	2009/tcp	
search	2010/tcp	
raid-cc	2011/tcp	raid
ttyinfo	2012/tcp	
raid-am	2013/tcp	
troff	2014/tcp	
cypress	2015/tcp	
cypress-stat	2017/tcp	
terminaldb	2018/tcp	
whosockami	2019/tcp	
servexec	2021/tcp	
down	2022/tcp	
ellpack	2025/tcp	
shadowserver	2027/tcp	
submitserver	2028/tcp	
device2	2030/tcp	
blackboard	2032/tcp	
glogger	2033/tcp	
scoremgr	2034/tcp	
imsldoc	2035/tcp	
objectmanager	2038/tcp	
lam	2040/tcp	
interbase	2041/tcp	
isis	2042/tcp	
rimsl	2044/tcp	
dls	2047/tcp	
dls-monitor	2048/tcp	
shilp	2049/tcp	
NSWS	3049/tcp	
rfa	4672/tcp	remote file access server
commplex-main	5000/tcp	
commplex-link	5001/tcp	
padl2sim	5236/tcp	
man	9535/tcp	

UDP

echo	7/udp	
discard	9/udp	sink null
systat	11/udp	users
daytime	13/udp	
netstat	15/udp	
qotd	17/udp	quote
chargen	19/udp	ttytst source
time	37/udp	timserver
rlp	39/udp	resource
name	42/udp	nameserver
whois	43/udp	nicname
nameserver	53/udp	domain
bootps	67/udp	bootp
bootpc	68/udp	
tftp	69/udp	
sunrpc	111/udp	
erpc	121/udp	
ntp	123/udp	

statsrv	133/udp	
profile	136/udp	
snmp	161/udp	
snmp-trap	162/udp	
at-rtmp	201/udp	
at-nbp	202/udp	
at-3	203/udp	
at-echo	204/udp	
at-5	205/udp	
at-zis	206/udp	
at-7	207/udp	
at-8	208/udp	
biff	512/udp	used by mail system to notify users of new mail received; currently receives messages only from processes on the same machine
who	513/udp	maintains data bases showing who's logged in to machines on a local net and the load average of the machine
syslog	514/udp	
talk	517/udp	like tenex link, but across machine - unfortunately, doesn't use link protocol (this is actually just a rendezvous port from which a tcp connection is established)
ntalk	518/udp	
utime	519/udp	unixtime
router	520/udp	local routing process (on site); uses variant of Xerox NS routing information protocol
timed	525/udp	timeserver
netwall	533/udp	for emergency broadcasts
new-rwho	550/udp	new-who
rmonitor	560/udp	rmonitord
monitor	561/udp	
meter	571/udp	udemon
elcsd	704/udp	errlog copy/server daemon
loadav	750/udp	
vid	769/udp	
cadlock	770/udp	
notify	773/udp	
acmaint_dbd	774/udp	
acmaint_transd	775/udp	
wpages	776/udp	
puparp	998/udp	
applix	999/udp	Applix ac
puprouter	999/udp	
cadlock	1000/udp	
hermes	1248/udp	
wizard	2001/udp	curry
globe	2002/udp	
emce	2004/udp	CCWS mm conf
oracle	2005/udp	
raid-cc	2006/udp	raid
raid-am	2007/udp	
terminaldb	2008/udp	
whosockami	2009/udp	
pipe_server	2010/udp	

servserv	2011/udp
raid-ac	2012/udp
raid-cd	2013/udp
raid-sf	2014/udp
raid-cs	2015/udp
bootserver	2016/udp
bootclient	2017/udp
rellpack	2018/udp
about	2019/udp
xinupageserver	2020/udp
xinuexpansion1	2021/udp
xinuexpansion2	2022/udp
xinuexpansion3	2023/udp
xinuexpansion4	2024/udp
xribs	2025/udp
scrabble	2026/udp
isis	2042/udp
isis-bcast	2043/udp
rimsl	2044/udp
cdfunc	2045/udp
sdfunc	2046/udp
dls	2047/udp
shilp	2049/udp
rmonitor_secure	5145/udp
xdsxdm	6558/udp
isode-dua	17007/udp

Internet Multicast Addresses

Host Extensions for IP Multicasting (RFC-1112) [43] specifies the extensions required of a host implementation of the Internet Protocol (IP) to support multicasting. Current addresses are listed below.

224.0.0.0	Reserved	[43,JBP]
224.0.0.1	All Hosts on this Subnet	[43,JBP]
224.0.0.2	All Gateways on this Subnet (prop)	[JBP]
224.0.0.3	Unassigned	[JBP]
224.0.0.4	DVMRP Routers	[140,JBP]
224.0.0.5	OSPFIGP All Routers	[83,JXM1]
224.0.0.6	OSPFIGP Designated Routers	[83,JXM1]
244.0.0.7-244.0.0.255	Unassigned	[JBP]
224.0.1.0	VMTP Managers Group	[17,DRC3]
224.0.1.1	NTP Network Time Protocol	[80,DLM1]
224.0.1.2	SGI-Dogfight	[AXC]
224.0.1.3	Rwhod	[SXD]
224.0.1.4	VNP	[DRC3]
244.0.1.5-244.0.1.255	Unassigned	[JBP]
224.0.2.1	"rwho" Group (BSD) (unofficial)	[JBP]
232.x.x.x	VMTP transient groups	[17,DRC3]

Note that when used on an Ethernet or IEEE 802 network, the 23 low-order bits of the IP Multicast address are placed in the low- order 23 bits of the Ethernet or IEEE 802 net multicast address 1.0.94.0.0.0. See the next section on "IANA ETHERNET ADDRESS BLOCK".

IANA Ethernet Address Block

The IANA owns an Ethernet address block which may be used for multicast address asignments or other special purposes.

The address block in IEEE binary is (which is in bit transmission order):

0000 0000 0000 0000 0111 1010

In the normal Internet dotted decimal notation this is 0.0.94 since the bytes are transmitted higher order first and bits within bytes are transmitted lower order first (see "Data Notation" in the Introduction).

IEEE CSMA/CD and Token Bus bit transmission order: 00 00 5E

IEEE Token Ring bit transmission order: 00 00 7A

Appearance on the wire (bits transmitted from left to right):

```
0                           23                              47
|                            |                               |
1000 0000 0000 0000 0111 1010 xxxx xxx0 xxxx xxxx xxxx xxxx
|                                      |
Multicast Bit                          0 = Internet Multicast
                                       1 = Assigned by IANA for
                                           other uses
```

Appearance in memory (bits transmitted right-to-left within octets, octets transmitted left-to-right):

```
0                           23                              47
|                            |                               |
0000 0001 0000 0000 0101 1110 0xxx xxxx xxxx xxxx xxxx xxxx
|                               |
Multicast Bit                   0 = Internet Multicast
                                1 = Assigned by IANA for
                                    other uses
```

The latter representation corresponds to the Internet standard bit- order, and is the format that most programmers have to deal with. Using this representation, the range of Internet Multicast addresses is:

01-00-5E-00-00-00	to	01-00-5E-7F-FF-FF	in hex, or
1.0.94.0.0.0	to	1.0.94.127.255.255	in dotted decimal

IP TOS Parameters

This documents the default Type-of-Service values that are currently recommended for the most important Internet protocols.

There are three binary TOS attributes: low delay, high throughput, and high reliability; in each case, an attribute bit is turned on to indicate "better". The three attributes cannot all be optimized simultanously, and in fact the TOS algorithms that have been discussed tend to make "better" values of the attributes mutually exclusive. Therefore, the recommended values have at most one bit on.

Generally, protocols which are involved in direct interaction with a human should select low delay, while data transfers which may involve large blocks of data are need high throughput. Finally, high reliability is most important for datagram-based Internet management functions.

Application protocols not included in these tables should be able to make appropriate choice of low delay (1 0 0) or high throughput (0 1 0).

The following are recommended values for TOS:

	----- Type-of-Service Value -----		
Protocol	**Low Delay**	**High Throughput**	**High Reliability**
TELNET (1)	1	0	0
FTP			
Control	1	0	0
Data (2)	0	1	0
TFTP	1	0	0
SMTP	(3)		
Cmd phase	1	0	0
DATA phase	0	1	0
Domain Name Service			
UDP Query	1	0	0
TCP Query	0	0	0
Zone Tnsfr	0	1	0
NNTP	0	0	0
ICMP			
Errors	0	0	0
Queries	0	0	0
Any IGP	0	0	1
EGP	0	0	0
SNMP	0	0	1

BOOTP	0	0	0

Notes:

(1) Includes all interactive user protocols (e.g., rlogin).

(2) Includes all bulk data transfer protocols (e.g., rcp).

(3) If the implementation does not support changing the TOS during the lifetime of the connection, then the recommended TOS on opening the connection is (0,0,0).

IP Time To Live Parameter

The current recommended default TTL for the Internet Protocol (IP) RFC-791 [45,105] is 32.

Domain System Parameters

The Internet Domain Naming System (DOMAIN) includes several parameters. These are documented in RFC-1034, [81] and RFC-1035 [82]. The CLASS parameter is listed here. The per CLASS parameters are defined in separate RFCs as indicated.

Domain System Parameters:

Decimal	Name	References
0	Reserved	[PM1]
1	Internet (IN)	[81,PM1]
2	Unassigned	[PM1]
3	Chaos (CH)	[PM1]
4	Hessoid (HS)	[PM1]
5-65534	Unassigned	[PM1]
65535	Reserved	

BOOTP Parameters

The Bootstrap Protocol (BOOTP) RFC-951 [36] describes an IP/UDP bootstrap protocol (BOOTP) which allows a diskless client machine to discover its own IP address, the address of a server host, and the name of a file to be loaded into memory and executed. The BOOTP Vendor Information Extensions RFC-1084 [117] proposes an addition to the Bootstrap Protocol (BOOTP).

Vendor Extensions are listed below:

Tag	Name	Data Length	Meaning
0	Pad	0	None
1	Subnet Mask	4	Subnet Mask Value
2	Time Zone	4	Time Offset in Seconds from UTC
3	Gateways	N	N/4 Gateway addresses
4	Time Server	N	N/4 Timeserver addresses
5	Name Server	N	N/4 IEN-116 Server addresses
6	Domain Server	N	N/4 DNS Server addresses
7	Log Server	N	N/4 Logging Server addresses
8	Quotes Server	N	N/4 Quotes Server addresses
9	LPR Server	N	N/4 Printer Server addresses
10	Impress Server	N	N/4 Impress Server addresses
11	RLP Server	N	N/4 RLP Server addresses
12	Hostname	N	Hostname string
13	Boot File Size	2	Size of boot file in 512 byte checks
14	Merit Dump File	Client to dump and name the file to dump it to	
15-127	Unassigned		
128-154	Reserved		
255	End	0	None

Network Management Parameters

For the management of hosts and gateways on the Internet a data structure for the information has been defined. This data structure should be used with any of several possible management protocols, such as the "Simple Network Management Protocol" (SNMP) RFC-1098 [15], or the "Common Management Information Protocol over TCP" (CMOT) [142].

The data structure is the "Structure and Indentification of Management Information for TCP/IP-based Internets" (SMI) RFC-1065 [120], and the "Management Information Base for Network Management of TCP/IP-based Internets" (MIB) [121].

The SMI includes the provision for parameters or codes to indicate experimental or private data structures. These parameter assignments are listed here.

The older "Simple Gateway Monitoring Protocol" (SGMP) RFC-1028 [37] also defined a data structure. The parameter assignments used with SGMP are included here for hist orical completeness.

SMI Network Management Experimental Codes:

Prefix: 1.3.6.1.3.

Decimal	Name	Description	References
0		Reserved	[JKR1]
1	CLNP	ISO CLNP Objects	[MTR]
2	T1-Carrier	T1 Carrier Objects	[MTR]
3	IEEE8023	Ethernet-like Objects	[MTR]
4	IEEE8025	Token Ring-like Objects	[MTR]

SMI Network Management Private Enterprise Codes:

Prefix: 1.3.6.1.4.1.

Decimal	Name	References
0	Reserved	[JKR1]
1	Proteon	[GSM11]
2	IBM	[JXR]
3	CMU	[SXW]
4	Unix	[KXS]
5	ACC	[AB20]
6	TWG	[KZM]
7	CAYMAN	[BP52]
8	NYSERNET	[MS9]
9	cisco	[GXS]
10	NSC	[GS123]
11	HP	[RDXS]
12	Epilogue	[KA4]
13	U of Tennessee	[JDC20]
14	BBN	[RH6]
15	Xylogics, Inc.	[JRL3]
16	Unisys	[UXW]
17	Canstar	[SXP]
18	Wellfleet	[JCB1]
19	TRW	[GGB2]
20	MIT	[JR35]

21	EON	[MXW]
22	Spartacus	[YXK]
23	Excelan	[RXB]
24	Spider Systems	[VXW]
25	NSFNET	[HWB]
26	Hughes LAN Systems	[AXC1]
27	Intergraph	[SXC]
28	Interlan	[FJK2]
29	Vitalink Communications	[FXB]
30	Ulana	[BXA]
31	NSWC	[SRN1]
32	Santa Cruz Operation	[KR35]
33	Xyplex	[BXS]
34	Cray	[HXE]
35	Bell Northern Research	[GXW]
36	DEC	[RXB1]
37	Touch	[BXB]
38	Network Research Corp.	[BXV]
39	Baylor College of Medicine	[SB98]
40	NMFECC-LLNL	[SXH]
41	SRI	[DW181]
42	Sun Microsystems	[DXY]
43	3Com	[TB6]
44	CMC	[DXP]
45	SynOptics	[BXB1]
46	Cheyenne Software	[RXH]
47	Prime Computer	[MXS]
48	MCNC/North Carolina Data Network	[KXW]
49	Chipcom	[JXC]
50	Optical Data Systems	[JXF]
51	gated	[JXH]
52	Cabletron Systems	[RXD]
53	Apollo Computers	[JXB]
54	DeskTalk Systems, Inc.	[DXK]
55	SSDS	[RXS]
56	Castle Rock Computing	[JXS1]
57	MIPS Computer Systems	[CXM]
58	TGV, Inc.	[KAA]
59	Silicon Graphics, Inc.	[RXJ]
60	University of British Columbia	[DXM]
61	Merit	[BXN]
62	FiberCom	[EXR]
63	Apple Computer Inc	[JXH1]
64	Gandalf	[HXK]
65	Dartmouth	[PXK]
66	David Systems	[DXM]
67	Reuter	[BXZ]
68	Cornell	[DC126]
69	TMAC	[MLS34]
70	Locus Computing Corp.	[AXS]
71	NASA	[SS92]
72	Retix	[AXM]
73	Boeing	[JXG]
74	AT&T	[AXC2]
75	Ungermann-Bass	[DXM]

76	Digital Analysis Corp.	[SXK]
77	LAN Manager	[JXG1]
78	Netlabs	[JB478]
79	ICL	[JXI]
80	Auspex Systems	[BXE]
81	Lannet Company	[EXR]
82	Network Computing Devices	[DM280]
83	Raycom Systems	[BXW1]
84	Pirelli Focom Ltd.	[SXL]
85	Datability Software Systems	[LXF]
86	Network Application Technology	[YXW]
87	LINK (Lokales Informatik-Netz Karlsruhe)	[GXS]
88	NYU	[BJR2]
89	RND	[RXN]
90	InterCon Systems Corporation	[AW90]

SGMP Vendor Specific Codes:

Prefix: 1,255,

Decimal	Name	References
0	Reserved	[JKR1]
1	Proteon	[JS18]
2	IBM	[JXR]
3	CMU	[SXW]
4	Unix	[MS9]
5	ACC	[AB20]
6	TWG	[MTR]
7	CAYMAN	[BP52]
8	NYSERNET	[MS9]
9	cisco	[GS2]
10	BBN	[RH6]
11	Unassigned	[JKR1]
12	MIT	[JR35]
13-254	Unassigned	[JKR1]
255	Reserved	[JKR1]

ARPANET and MILNET Logical Addresses

The ARPANET facility for "logical addressing" is described in RFC-878 [57] and RFC-1005 [109]. A portion of the possible logical addresses are reserved for standard uses.

There are 49,152 possible logical host addresses. Of these, 256 are reserved for assignment to well-known functions. Assignments for well-known functions are made by the IANA. Assignments for other logical host addresses are made by the NIC.

Logical Address Assignments:

Decimal	Description	References
0	Reserved	[JBP]
1	The BBN Core Gateways	[MB]
2-254	Unassigned	[JBP]
255	Reserved	[JBP]

ARPANET and MILNET Link Numbers

The word "link" here refers to a field in the original ARPANET Host/IMP interface leader. The link was originally defined as an 8- bit field. Later specifications defined this field as the "message- id" with a length of 12 bits. The name link now refers to the high order 8 bits of this 12-bit message-id field. The Host/IMP interface is defined in BBN Report 1822 [2].

The low-order 4 bits of the message-id field are called the sub-link. Unless explicitly specified otherwise for a particular protocol, there is no sender to receiver significance to the sub-link. The sender may use the sub-link in any way he chooses (it is returned in the RFNM by the destination IMP), the receiver should ignore the sub-link.

Link Assignments:

Decimal	Description	References
0-63	BBNCC Monitoring	[MB]
64-149	Unassigned	[JBP]
150	Xerox NS IDP	[133,XEROX]
151	Unassigned	[JBP]
152	PARC Universal Protocol	[8,XEROX]
153	TIP Status Reporting	[JGH]
154	TIP Accounting	[JGH]
155	Internet Protocol [regular]	[105,JBP]
156-158	Internet Protocol [exper]	[105,JBP]
159	Figleaf Link	[JBW1]
160	Blacker Local Network	[DM28]
161-194	Unassigned	[JBP]
195	ISO-IP	[64,RXM]
196-247	Experimental Protocols	[JBP]
248-255	Network Maintenance	[JGH]

ARPANET and MILNET X.25 Address Mappings

All MILNET hosts are assigned addresses by the Defense Data Network (DDN). The address of a MILNET host may be obtained from the Network Information Center (NIC), represented as an ASCII text string in what is called "host table format". This section describes the process by which MILNET X.25 addresses may be derived from addresses in the NIC host table format.

A NIC host table address consists of the ASCII text string representations of four decimal numbers separated by periods, corresponding to the four octeted of a thirty-two bit Internet address. The four decimal numbers are referred to in this section as "n", "h' "l", and "i". Thus, a host table address may be represented as: "n.h.l.i". Each of these four numbers will have either one, two, or three decimal digits and will never have a value greater than 255. For example, in the host table, address: "10.2.0.124", n=10, h=2, l=0, and i=124. To convert a host table address to a MILNET X.25 address:

1. If h < 64, the host table address corresponds to the X.25 physical address:

   ZZZZ F IIIHHZZ (SS)

   where:

   ZZZZ = 0000 as required

   F = 0 because the address is a physical address;

   III is a three decimal digit respresentation of "i", right-adjusted and padded with leading zeros if required;

   HH is a two decimal digit representation of "h", right-adjusted and padded with leading zeros if required;

   ZZ = 00 and

   (SS) is optional

   In the example given above, the host table address 10.2.0.124 corresponds to the X.25 physical address 000001240200.

2. If h > 64 or h = 64, the host table address corresponds to the X.25 logical address

   ZZZZ F RRRRRZZ (SS)

   where:

   ZZZZ = 0000 as required

   F = 1 because the address is a logical address;

   RRRRR is a five decimal digit representation of the result "r" of the calculation

   r = h * 256 + i

   (Note that the decimal representation of "r" will always require five digits);

   ZZ = 00 and

   (SS) is optional

   Thus, the host table address 10.83.0.207 corresponds to the X.25 logical address 000012145500.

In both cases, the "n" and "l" fields of the host table address are not used.

IEEE 802 Numbers Of Interest

Some of the networks of all classes are IEEE 802 Networks. These systems may use a Link Service Access Point (LSAP) field in much the same way the ARPANET uses the "link" field. Further, there is an extension of the LSAP header called the Sub-Network Access Protocol (SNAP).

The IEEE likes to describe numbers in binary in bit transmission order, which is the opposite of the big-endian order used throughout the Internet protocol documentation.

Assignments:

Link Service Access Point			Description	References
IEEE binary	Internet binary	decimal		
00000000	00000000	0	Null LSAP	[IEEE]
01000000	00000010	2	Indiv LLC Sublayer Mgt	[IEEE]
11000000	00000011	3	Group LLC Sublayer Mgt	[IEEE]
00100000	00000100	4	SNA Path Control	[IEEE]
01100000	00000110	6	Reserved (DOD IP)	[104,JBP]
01110000	00001110	14	PROWAY-LAN	[IEEE]
01110010	01001110	78	EIA-RS 511	[IEEE]
01111010	01011110	94	ISI IP	[JBP]
01110001	10001110	142	PROWAY-LAN	[IEEE]
01010101	10101010	170	SNAP	[IEEE]
01111111	11111110	254	ISO DIS 8473	[64,JXJ]
11111111	11111111	255	Global DSAP	[IEEE]

These numbers (and others) are assigned by the IEEE Standards Office. The address is: IEEE Standards Office, 345 East 47th Street, New York, N.Y. 10017, Attn: Vince Condello. Phone: (212) 705-7092.

At an ad hoc special session on "IEEE 802 Networks and ARP", held during the TCP Vendors Workshop (August 1986), an approach to a consistent way to send DoD-IP datagrams and other IP related protocols (such as the Address Resolution Protocol (ARP)) on 802 networks was developed, using the SNAP extension (see RFC-1010 and RFC-1042 [90]).

ETHERNET Numbers Of Interest

Many of the networks of all classes are Ethernets (10Mb) or Experimental Ethernets (3Mb). These systems use a message "type" field in much the same way the ARPANET uses the "link" field.

If you need an Ethernet type, contact the Xerox Corporation, Xerox Systems Institute, 475 Oakmead Parkway, Sunnyvale, CA 94086, Attn: Ms. Fonda Pallone, (408) 737-4652.

The following list is contributed unverified information from various sources.

Assignments:

Ethernet decimal	Hex	Exp. Ethernet decimal	octal	Description	References
000	0000-05DC	-	-	IEEE802.3 Length Field	[XEROX]
257	0101-01FF	-	-	Experimental	[XEROX]
512	0200	512	1000	XEROX PUP (0A00)	[8,XEROX]
513	0201	-	-	PUP Addr Trans (0A01)	[XEROX]
1536	0600	1536	3000	XEROX NS IDP	[133,XEROX]
2048	0800	513	1001	DOD IP	[105,JBP]
2049	0801	-	-	X.75 Internet	[XEROX]
2050	0802	-	-	NBS Internet	[XEROX]
2051	0803	-	-	ECMA Internet	[XEROX]
2052	0804	-	-	Chaosnet	[XEROX]
2053	0805	-	-	X.25 Level 3	[XEROX]
2054	0806	-	-	ARP	[88,JBP]
2055	0807	-	-	XNS Compatability	[XEROX]
2076	081C	-	-	Symbolics Private	[DCP1]
2184	0888-088A	-	-	Xyplex	[XEROX]
2304	0900	-	-	Ungermann-Bass debug	[XEROX]
2560	0A00	-	-	Xerox IEEE802.3 PUP	[XEROX]
2561	0A01	-	-	PUP Addr Trans	[XEROX]
2989	0BAD	-	-	Banyan Systems	[XEROX]
4096	1000	-	-	Berkeley Trailer nego	[XEROX]
4097	1001-100F	-	-	Berkeley Trailer encap	[XEROX]
5632	1600	-	-	Valid Systems	[XEROX]
16962	4242	-	-	PCS Basic Block	[XEROX]
21000	5208	-	-	BBN Simnet	[XEROX]
24576	6000	-	-	DEC Unassigned (Exp.)	[XEROX]
24577	6001	-	-	DEC MOP Dump/Load	[XEROX]
24578	6002	-	-	DEC MOP Remote Con.	[XEROX]
24579	6003	-	-	DEC DECNET Phase IV	[XEROX]
24580	6004	-	-	DEC LAT	[XEROX]
24581	6005	-	-	DEC Diagnostic Protocol	[XEROX]
24582	6006	-	-	DEC Customer Protocol	[XEROX]
24583	6007	-	-	DEC LAVC, SCA	[XEROX]
24584	6008-6009	-	-	DEC Unassigned	[XEROX]
24586	6010-6014	-	-	3Com Corporation	[XEROX]

28672	7000	-	-	Ungermann-Bass dnld	[XEROX]
28674	7002	-	-	Ungermann-Bass dia/lp	[XEROX]
28704	7020-7029	-	-	LRT	[XEROX]
28720	7030	-	-	Proteon	[XEROX]
28724	7034	-	-	Cabletron	[XEROX]
32771	8003	-	-	Cronus VLN	[131,DT15]
32772	8004	-	-	Cronus Direct	[131,DT15]
32773	8005	-	-	HP Probe	[XEROX]
32774	8006	-	-	Nestar	[XEROX]
32776	8008	-	-	AT&T	[XEROX]
32784	8010	-	-	Excelan	[XEROX]
32787	8013	-	-	SGI diagnostics	[AXC]
32788	8014	-	-	SGI network games	[AXC]
32789	8015	-	-	SGI reserved	[AXC]
32780	8016	-	-	SGI bounce server	[AXC]
32783	8019	-	-	Apollo Computers	[XEROX]
32815	802E	-	-	Tymshare	[XEROX]
32816	802F	-	-	Tigan, Inc.	[XEROX]
32821	8035	-	-	Reverse ARP	[48,JXM]
32822	8036	-	-	Aeonic Systems	[XEROX]
32824	8038	-	-	DEC LANBridge	[XEROX]
32825	8039-803C	-	-	DEC Unassigned	[XEROX]
32829	803D	-	-	DEC Ethernet Encryption	[XEROX]
32830	803E	-	-	DEC Unassigned	[XEROX]
32831	803F	-	-	DEC LAN Traffic Monitor	[XEROX]
32832	8040-8042	-	-	DEC Unassigned	[XEROX]
32836	8044	-	-	Planning Research Corp	[XEROX]
32838	8046	-	-	AT&T	[XEROX]
32839	8047	-	-	AT&T	[XEROX]
32841	8049	-	-	ExperData	[XEROX]
32859	805B	-	-	Stanford V Kernel exp.	[XEROX]
32860	805C	-	-	Stanford V Kernel prod.	[XEROX]
32861	805D	-	-	Evans & Sutherland	[XEROX]
32864	8060	-	-	Little Machines	[XEROX]
32866	8062	-	-	Counterpoint Computers	[XEROX]
32869	8065-8066	-	-	Univ. of Mass. @Amherst	[XEROX]
32871	8067	-	-	Veeco Integrated Auto.	[XEROX]
32872	8068	-	-	General Dynamics	[XEROX]
32873	8069	-	-	AT&T	[XEROX]
32874	806A	-	-	Autophon	[XEROX]
32876	806C	-	-	ComDesign	[XEROX]
32877	806D	-	-	Computgraphic Corp.	[XEROX]
32878	806E-8077	-	-	Landmark Graphics Cor	[XEROX]
32890	807A	-	-	Matra	[XEROX]
32891	807B	-	-	Dansk Data Elektronik	[XEROX]
32892	807C	-	-	Merit Internodal	[HWB]
32893	807D-807F	-	-	Vitalink Communications	[XEROX]
32896	8080	-	-	Vitalink TransLAN III	[XEROX]
32897	8081-8083	-	-	Counterpoint Computers	[XEROX]

32923	809B	-	-	Appletalk	[XEROX]
32924	809C-809E	-	-	Datability	[XEROX]
32927	809F	-	-	Spider Systems Ltd.	[XEROX]
32931	80A3	-	-	Nixdorf Computers	[XEROX]
32932	80A4-80B3	-	-	Siemens Gammasonics	[XEROX]
32960	80C0-80C3	-	-	DCA Data Exch Cluster	[XEROX]
32966	80C6	-	-	Pacer Software	[XEROX]
32967	80C7	-	-	Applitek Corporation	[XEROX]
32968	80C8-80CC	-	-	Intergraph Corporation	[XEROX]
32973	80CD-80CE	-	-	Harris Corporation	[XEROX]
32974	80CF-80D2	-	-	Taylor Instrument	[XEROX]
32979	80D3-80D4	-	-	Rosemount Corporation	[XEROX]
32981	80D5	-	-	IBM SNA Serv on Ether	[XEROX]
32989	80DD	-	-	Varian Associates	[XEROX]
32990	80DE-80DF	-	-	Integrated Solutions	[XEROX]
32992	80E0-80E3	-	-	Allen-Bradley	[XEROX]
32996	80E4-80F0	-	-	Datability	[XEROX]
33010	80F2	-	-	Retix	[XEROX]
33011	80F3	-	-	AppleTalk AARP	[XEROX]
33012	80F4-80F5	-	-	Kinetics	[XEROX]
33015	80F7	-	-	Apollo Computer	[XEROX]
33023	80FF-8103	-	-	Wellfleet Commun	[XEROX]
33031	8107-8109	-	-	Symbolics Private	[XEROX]
33072	8130	-	-	Waterloo Microsystems	[XEROX]
33073	8131	-	-	VG Laboratory Systems	[XEROX]
33079	8137-8138	-	-	Novell, Inc.	[XEROX]
33081	8139-813D	-	-	KTI	[XEROX]
33100	814C	-	-	SNMP	[JKR1]
36864	9000	-	-	Loopback	[XEROX]
36865	9001	-	-	3Com(Bridge) XNS Mgt	[XEROX]
36866	9002	-	-	3Com(Bridge) TCP-IP	[XEROX]
36867	9003	-	-	3Com(Bridge) lp detect	[XEROX]
65280	FF00	-	-	BBN VITAL-LanBridge	[XEROX]

The standard for transmission of IP datagrams over Ethernets and Experimental Ethernets is specified in RFC-894 [61] and RFC-895 [91] respectively.

NOTE: Ethernet 48-bit address blocks are assigned by the IEEE.

IEEE Standards Office, 345 East 47th Street, New York, N.Y. 10017,

Attn: Vince Condello. Phone: (212) 705-7092.

Ethernet Vendor Address Components

Ethernet hardware addresses are 48 bits, expressed as 12 hexadecimal digits (0-9, plus A-F, capitalized). These 12 hex digits consist of the first/left 6 digits (which should match the vendor of the Ethernet interface within the station) and the last/right 6 digits which specify the interface serial number for that interface vendor.

Ethernet addresses might be written unhyphenated (e.g., 123456789ABC), or with one hyphen (e.g., 123456-789ABC), but should be written hyphenated by octets (e.g., 12-34-56-78-9A-BC).

These addresses are physical station addresses, not multicast nor broadcast, so the second hex digit (reading from the left) will be even, not odd.

At present, it is not clear how the IEEE assigns Ethernet block addresses. Whether in blocks of 2**24 or 2**25, and whether multicasts are assigned with that block or separately. A portion of the vendor block address is reportedly assigned serially, with the other portion intentionally assigned randomly. If there is a global algorithm for which addresses are designated to be physical (in a chipset) versus logical (assigned in software), or globally-assigned versus locally-assigned addresses, some of the known addresses do not follow the scheme (e.g., AA0003; 02xxxx).

00000C	Cisco
00000F	NeXT
000010	Sytek
00001D	Cabletron
000020	DIAB (Data Intdustrier AB)
000022	Visual Technology
00002A	TRW
00005A	S & Koch
00005E	IANA
000065	Network General
00006B	MIPS
000077	MIPS
00007A	Ardent
000089	Cayman Systems Gatorbox
000093	Proteon
00009F	Ameristar Technology
0000A2	Wellfleet
0000A3	Network Application Technology
0000A6	Network General (internal assignment, not for products)
0000A7	NCD X-terminals
0000A9	Network Systems
0000AA	Xerox Xerox machines
0000B3	CIMLinc
0000B7	Dove Fastnet
0000BC	Allen-Bradley
0000C0	Western Digital
0000C6	HP Intelligent Networks Operation (formerly Eon Systems)
0000C8	Altos
0000C9	Emulex Terminal Servers
0000D7	Dartmouth College (NED Router)
0000D8	3Com? Novell? PS/2
0000DD	Gould

0000DE	Unigraph
0000E2	Acer Counterpoint
0000EF	Alantec
0000FD	High Level Hardvare (Orion, UK)
000102	BBN BBN internal usage (not registered)
001700	Kabel
00802D	Xylogics, Inc. Annex terminal servers
00808C	Frontier Software Development
00AA00	Intel
00DD00	Ungermann-Bass
00DD01	Ungermann-Bass
020701	MICOM/Interlan UNIBUS or QBUS machines, Apollo
020406	BBN BBN internal usage (not registered)
026086	Satelcom MegaPac (UK)
02608C	3Com IBM PC; Imagen; Valid; Cisco
02CF1F	CMC Masscomp; Silicon Graphics; Prime EXL
080002	3Com (Formerly Bridge)
080003	ACC (Advanced Computer Communications)
080005	Symbolics Symbolics LISP machines
080008	BBN
080009	Hewlett-Packard
08000A	Nestar Systems
08000B	Unisys
080010	AT&T
080011	Tektronix, Inc.
080014	Excelan BBN Butterfly, Masscomp, Silicon Graphics
080017	NSC
08001A	Data General
08001B	Data General
08001E	Apollo
080020	Sun Sun machines
080022	NBI
080025	CDC
080026	Norsk Data (Nord)
080027	PCS Computer Systems GmbH
080028	TI Explorer
08002B	DEC
08002E	Metaphor
08002F	Prime Computer Prime 50-Series LHC300
080036	Intergraph CAE stations
080037	Fujitsu-Xerox
080038	Bull
080039	Spider Systems
080041	DCA Digital Comm. Assoc.
080045	???? (maybe Xylogics, but they claim not to know this number)
080046	Sony
080047	Sequent
080049	Univation
08004C	Encore
08004E	BICC
080056	Stanford University
080058	??? DECsystem-20
08005A	IBM
080067	Comdesign
080068	Ridge

080069	Silicon Graphics
08006E	Excelan
080075	DDE (Danish Data Elektronik A/S)
08007C	Vitalink TransLAN III
080080	XIOS
080086	Imagen/QMS
080087	Xyplex terminal servers
080089	Kinetics AppleTalk-Ethernet interface
08008B	Pyramid
08008D	XyVision XyVision machines
080090	Retix Inc Bridges
484453	HDS ???
800010	AT&T [misrepresentation of 080010?]
AA0000	DEC obsolete
AA0001	DEC obsolete
AA0002	DEC obsolete
AA0003	DEC Global physical address for some DEC machines
AA0004	DEC Local logical address for systems running DECNET

Ethernet Multicast Addresses

Ethernet Address	Type Field	Usage
Multicast Addresses:		
01-00-5E-00-00-00- 01-00-5E-7F-FF-FF	0800	Internet Multicast (RFC-1112) [43]
01-00-5E-80-00-00- 01-00-5E-FF-FF-FF	????	Internet reserved by IANA
01-80-C2-00-00-00	-802-	Spanning tree (for bridges)
09-00-02-04-00-01?	8080?	Vitalink printer
09-00-02-04-00-02?	8080?	Vitalink management
09-00-09-00-00-01	8005	HP Probe
09-00-09-00-00-01	-802-	HP Probe
09-00-09-00-00-04	8005?	HP DTC
09-00-1E-00-00-00	8019?	Apollo DOMAIN
09-00-2B-00-00-00	6009?	DEC MUMPS?
09-00-2B-00-00-01	8039?	DEC DSM/DTP?
09-00-2B-00-00-02	803B?	DEC VAXELN?
09-00-2B-00-00-03	8038	DEC Lanbridge Traffic Monitor (LTM)
09-00-2B-00-00-04	????	DEC MAP End System Hello?
09-00-2B-00-00-05	????	DEC MAP Intermediate System Hello?
09-00-2B-00-00-06	803D?	DEC CSMA/CD Encryption?
09-00-2B-00-00-07	8040?	DEC NetBios Emulator?
09-00-2B-00-00-0F	6004	DEC Local Area Transport (LAT)
09-00-2B-00-00-1x	????	DEC Experimental
09-00-2B-01-00-00	8038	DEC LanBridge Copy packets (All bridges)
09-00-2B-01-00-01	8038	DEC LanBridge Hello packets (All local bridges) 1 packet per second, sent by the designated LanBridge
09-00-2B-02-00-00	????	DEC DNA Level 2 Routing Layer routers?
09-00-2B-02-01-00	803C?	DEC DNA Naming Service Advertisement?
09-00-2B-02-01-01	803C?	DEC DNA Naming Service Solicitation?
09-00-2B-02-01-02	803E?	DEC DNA Time Service?
09-00-2B-03-xx-xx	????	DEC default filtering by bridges?
09-00-2B-04-00-00	8041?	DEC Local Area System Transport (LAST)?
09-00-2B-23-00-00	803A?	DEC Argonaut Console?
09-00-4E-00-00-02?	8137?	Novell IPX
09-00-56-00-00-00- 09-00-56-FE-FF-FF	????	Stanford reserved
09-00-56-FF-00-00- 09-00-56-FF-FF-FF	805C	Stanford V Kernel, version 6.0
09-00-77-00-00-01	????	Retix spanning tree bridges
09-00-7C-02-00-05	8080?	Vitalink diagnostics
09-00-7C-05-00-01	8080?	Vitalink gateway?
0D-1E-15-BA-DD-06	????	HP
AB-00-00-01-00-00	6001	DEC Maintenance Operation Protocol (MOP) Dump/Load Assistance
AB-00-00-02-00-00	6002	DEC Maintenance Operation Protocol (MOP) Remote Console 1 System ID packet every 8-10 minutes, by every: DEC LanBridge

		DEC DEUNA interface DEC DELUA interface DEC DEQNA interface (in a certain mode)
AB-00-00-03-00-00	6003	DECNET Phase IV end node Hello packets 1 packet every 15 seconds, sent by each DECNET host
AB-00-00-04-00-00	6003	DECNET Phase IV Router Hello packets 1 packet every 15 seconds, sent by the DECNET router
AB-00-00-05-00-00 through AB-00-03-FF-FF-FF	????	Reserved DEC
AB-00-03-00-00-00	6004	DEC Local Area Transport (LAT) - old
AB-00-04-00-xx-xx	????	Reserved DEC customer private use
AB-00-04-01-xx-yy	6007	DEC Local Area VAX Cluster groups System Communication Architecture (SCA)
CF-00-00-00-00-00	9000	Ethernet Configuration Test protocol (Loopback)

Broadcast Address:

FF-FF-FF-FF-FF-FF	0600	XNS packets, Hello or gateway search? 6 packets every 15 seconds, per XNS station
FF-FF-FF-FF-FF-FF	0800	IP (e.g. RWHOD via UDP) as needed
FF-FF-FF-FF-FF-FF	0804	CHAOS
FF-FF-FF-FF-FF-FF	0806	ARP (for IP and CHAOS) as needed
FF-FF-FF-FF-FF-FF	0BAD	Banyan
FF-FF-FF-FF-FF-FF	1600	VALID packets, Hello or gateway search? 1 packets every 30 seconds, per VALID station
FF-FF-FF-FF-FF-FF	8035	Reverse ARP
FF-FF-FF-FF-FF-FF	807C	Merit Internodal (INP)
FF-FF-FF-FF-FF-FF	809B	EtherTalk

XNS Protocol Types

Assigned well-known socket numbers

Routing Information	1
Echo	2
Router Error	3
Experimental	40-77

Assigned internet packet types

Routing Information	1
Echo	2
Error	3
Packet Exchange	4
Sequenced Packet	5
PUP	12
DoD IP	13
Experimental	20-37

Protocol/Type Field Assignments

Below are two tables describing the arrangement of protocol fields or type field assignments so that one could send NS Datagrams on the ARPANET or Internet Datagrams on 10Mb Ethernet, and also protocol and type fields so one could encapsulate each kind of Datagram in the other.

\ upper / lower \	DoD IP	PUP	NS IP
3Mb Ethernet	Type 1001 octal	Type 1000 octal	Type 3000 octal
10 Mb Ethernet	Type 0800 hex	Type 0200 hex	Type 0600 hex
ARPANET	Link 155 decimal	Link 152 decimal	Link 150 decimal

\ upper / lower \	DoD IP	PUP	NS IP
DoD IP	X	Protocol 12 decimal	Protocol 22 decimal
PUP	?	X	?
NS IP	Type 13 decimal	Type 12 decimal	X

PRONET 80 Type Numbers

Below is the current list of PRONET 80 Type Numbers. Note: a protocol that is on this list does not necessarily mean that there is any implementation of it on ProNET.

Of these, protocols 1, 14, and 20 are the only ones that have ever been seen in ARP packets.

For reference, the header is (one byte/line):

destination hardware address
source hardware address
data link header version (2)
data link header protocol number
data link header reserved (0)
data link header reserved (0)

Some protocols have been known to tuck stuff in the reserved fields.

Those who need a protocol number on ProNET-10/80 should contact John Shriver (jas@proteon.com).

1	IP
2	IP with trailing headers
3	Address Resoloution Protocol
4	Proteon HDLC
5	VAX Debugging Protocol (MIT)
10	Novell NetWare (IPX and pre-IPX) (old format, 3 byte trailer)
11	Vianetix
12	PUP
13	Watstar protocol (University of Waterloo)
14	XNS
15	Diganostics
16	Echo protocol (link level)
17	Banyan Vines
20	DECnet (DEUNA Emulation)
21	Chaosnet
23	IEEE 802.2 or ISO 8802/2 Data Link
24	Reverse Address Resolution Protocol
29	TokenVIEW-10
31	AppleTalk LAP Data Packet
33	Cornell Boot Server Location Protocol
34	Novell NetWare IPX (new format, no trailer, new XOR checksum)

Address Resolution Protocol Parameters

The Address Resolution Protocol (ARP) specified in RFC-826 [88] has several parameters. The assigned values for these parameters are listed here.

Assignments:

Operation Code (op)

1 REQUEST
2 REPLY

Hardware Type (hrd)

Type	Description	References
1	Ethernet (10Mb)	[JBP]
2	Experimental Ethernet (3Mb)	[JBP]
3	Amateur Radio AX.25	[PXK]
4	Proteon ProNET Token Ring	[JBP]
5	Chaos	[GXP]
6	IEEE 802 Networks	[JBP]
7	ARCNET	[JBP]
8	Hyperchannel	[JBP]
9	Lanstar	[TU]
10	Autonet Short Address	[MXB1]
11	LocalTalk	[LXE]
12	LocalNet (IBM PCNet or SYTEK LocalNET)	[JXM]

Protocol Type (pro)

Use the same codes as listed in the section called "Ethernet Numbers of Interest" (all hardware types use this code set for the protocol type).

Reverse Address Resolution Protocol Operation Codes

The Reverse Address Resolution Protocol (RARP) specified in RFC-903 [48] has the following operation codes:

Assignments:

Operation Code (op)
3 request Reverse
4 reply Reverse

Dynamic Reverse Arp

Assignments:

Operation Code (op)
5 DRARP-Request
6 DRARP-Reply
7 DRARP-Error

For further information, contact: David Brownell (suneast!helium!db@Sun.COM).

X.25 Type Numbers

CCITT defines the high order two bits of the first octet of call user data as follows:

00 - Used for other CCITT recomendations (such as X.29)
01 - Reserved for use by "national" administrative authorities
10 - Reserved for use by international administrative authoorities
11 - Reserved for arbitrary use between consenting DTEs

Call User Data (hex)		Protocol	Reference
01	PAD		[GS2]
C5	Blacker front-end descr dev		[AGM]
CC	IP		[69,AGM]*
CD	ISO-IP		[AGM]

* **NOTE:** ISO SC6/WG2 approved assignment in ISO 9577 (January 1990).

Public Data Network Numbers

One of the Internet Class A Networks is the international system of Public Data Networks. This section lists the mapping between the Internet Addresses and the Public Data Network Addresses (X.121).

The numbers below are assigned for networks that are connected to the Internet, and for independent networks. These independent networks are marked with an asterisk preceding the number.

Assignments:

* Internet	Public Data Net	Description	References
014.000.000.000			Reserved [JBP]
014.000.000.001		3110-317-00035 00	PURDUE-TN [TN]
014.000.000.002		3110-608-00027 00	UWISC-TN [TN]
014.000.000.003		3110-302-00024 00	UDEL-TN [TN]
014.000.000.004		2342-192-00149 23	UCL-VTEST [PK]
014.000.000.005		2342-192-00300 23	UCL-TG [PK]
014.000.000.006		2342-192-00300 25	UK-SATNET [PK]
014.000.000.007		3110-608-00024 00	UWISC-IBM [MS56]
014.000.000.008		3110-213-00045 00	RAND-TN [MO2]
014.000.000.009		2342-192-00300 23	UCL-CS [PK]
014.000.000.010		3110-617-00025 00	BBN-VAN-GW [JD21]
*014.000.000.011		2405-015-50300 00	CHALMERS [UXB]
014.000.000.012		3110-713-00165 00	RICE [PAM6]
014.000.000.013		3110-415-00261 00	DECWRL [PAM6]
014.000.000.014		3110-408-00051 00	IBM-SJ [SA1]
014.000.000.015		2041-117-01000 00	SHAPE [JFW]
014.000.000.016		2628-153-90075 00	DFVLR4-X25 [GB7]
014.000.000.017		3110-213-00032 00	ISI-VAN-GW [JD21]
014.000.000.018		2624-522-80900 52	FGAN-SIEMENS-X25 [GB7]
014.000.000.019		2041-170-10000 00	SHAPE-X25 [JFW]
014.000.000.020		5052-737-20000 50	UQNET [AXH]
014.000.000.021		3020-801-00057 50	DMC-CRC1 [VXT]
014.000.000.022		2624-522-80329 02	FGAN-FGANFFMVAX-X25 [GB7]
*014.000.000.023		2624-589-00908 01	ECRC-X25 [PXD]
014.000.000.024		2342-905-24242 83	UK-MOD-RSRE [JXE2]
014.000.000.025		2342-905-24242 82	UK-VAN-RSRE [AXM]
014.000.000.026		2624-522-80329 05	DFVLRSUN-X25 [GB7]
014.000.000.027		2624-457-11015 90	SELETFMSUN-X25 [BXD]
014.000.000.028		3110-408-00146 00	CDC-SVL [RAM57]
014.000.000.029		2222-551-04400 00	SUN-CNUCE [ABB2]
014.000.000.030		2222-551-04500 00	ICNUCEVM-CNUCE [ABB2]
014.000.000.031		2222-551-04600 00	SPARE-CNUCE [ABB2]
014.000.000.032		2222-551-04700 00	ICNUCEVX-CNUCE [ABB2]
014.000.000.033		2222-551-04524 00	CISCO-CNUCE [ABB2]
014.000.000.034		2342-313-00260 90	SPIDER-GW [AD67]
014.000.000.035		2342-313-00260 91	SPIDER-EXP [AD67]
014.000.000.036		2342-225-00101 22	PRAXIS-X25A [TXR]
014.000.000.037		2342-225-00101 23	PRAXIS-X25B [TXR]
014.000.000.038		2403-712-30250 00	DIAB-TABY-GW [FXB]

014.000.000.039	2403-715-30100 00	DIAB-LKP-GW [FXB]
014.000.000.040	2401-881-24038 00	DIAB-TABY1-GW [FXB]
014.000.000.041	2041-170-10060 00	STC [TC27]
014.000.000.042-014.255.255.254	Unassigned	[JBP]
014.255.255.255		Reserved [JBP]

The standard for transmission of IP datagrams over the Public Data Network is specified in RFC-877 [69].

Telnet Options

The Telnet Protocol has a number of options that may be negotiated. These options are listed here. "Official Internet Protocols" [118] provides more detailed information.

Options	Name	References
0	Binary Transmission	[110,JBP]
1	Echo	[111,JBP]
2	Reconnection	[42,JBP]
3	Suppress Go Ahead	[114,JBP]
4	Approx Message Size Negotiation	[133,JBP]
5	Status	[113,JBP]
6	Timing Mark	[115,JBP]
7	Remote Controlled Trans and Echo	[107,JBP]
8	Output Line Width	[40,JBP]
9	Output Page Size	[41,JBP]
10	Output Carriage-Return Disposition	[28,JBP]
11	Output Horizontal Tab Stops	[32,JBP]
12	Output Horizontal Tab Disposition	[31,JBP]
13	Output Formfeed Disposition	[29,JBP]
14	Output Vertical Tabstops	[34,JBP]
15	Output Vertical Tab Disposition	[33,JBP]
16	Output Linefeed Disposition	[30,JBP]
17	Extended ASCII	[136,JBP]
18	Logout	[25,MRC]
19	Byte Macro	[35,JBP]
20	Data Entry Terminal	[145,38,JBP]
21	SUPDUP	[26,27,MRC]
22	SUPDUP Output	[51,MRC]
23	Send Location	[68,EAK1]
24	Terminal Type	[128,MS56]
25	End of Record	[103,JBP]
26	TACACS User Identification	[1,BA4]
27	Output Marking	[125,SXS]
28	Terminal Location Number	[84,RN6]
29	Telnet 3270 Regime	[116,JXR]
30	X.3 PAD	[70,SL70]
31	Negotiate About Window Size	[139,DW183]
32	Terminal Speed	[57,CLH3]
33	Remote Flow Control	[58,CLH3]
34	Linemode	[9,DB14]
35	X Display Location	[75,GM23]
255	Extended-Options-List	[109,JBP]

Mail Encryption Types

RFC-822 specifies that Encryption Types for mail may be assigned. There are currently no RFC-822 encryption types assigned. Please use instead the Mail Privacy procedures defined in [71,72,66].

Machine Names

These are the Official Machine Names as they appear in the Domain Name System WKS records and the NIC Host Table. Their use is described in RFC-952 [53].

A machine name or CPU type may be up to 40 characters taken from the set of uppercase letters, digits, and the two punctuation characters hyphen and slash. It must start with a letter, and end with a letter or digit.

ALTO	ALTOS-6800	AMDAHL-V7
APOLLO	ATARI-104ST	ATT-3B1
ATT-3B20	ATT-7300	BBN-C/60
BURROUGHS-B/29	BURROUGHS-B/4800	BUTTERFLY
C/30	C/70	CADLINC
CADR	CDC-170	CDC-170/750
CDC-173	CELERITY-1200	CLUB-386
COMPAQ-386/20	COMTEN-3690	CP8040
CRAY-1	CRAY-2	CRAY-X/MP
CTIWS-117	DANDELION	DEC-10
DEC-1050	DEC-1077	DEC-1080
DEC-1090	DEC-1090B	DEC-1090T
DEC-2020T	DEC-2040	DEC-2040T
DEC-2050T	DEC-2060	DEC-2060T
DEC-2065	DEC-FALCON	DEC-KS10
DEC-VAX-11730	DORADO	DPS8/70M
ELXSI-6400	EVEREX-386	FOONLY-F2
FOONLY-F3	FOONLY-F4	GOULD
GOULD-6050	GOULD-6080	GOULD-9050
GOULD-9080	H-316	H-60/68
H-68	H-68/80	H-89
HONEYWELL-DPS-6	HONEYWELL-DPS-8/70	HP3000
HP3000/64	IBM-158	IBM-3081
IBM-3084QX	IBM-3101	IBM-360/67
IBM-370/3033	IBM-4331	IBM-4341
IBM-4361	IBM-4381	IBM-4956
IBM-6152	IBM-PC	IBM-PC/AT
IBM-PC/RT	IBM-PC/XT	IBM-SERIES/1
IMAGEN	IMAGEN-8/300	IMSAI
INTEGRATED-SOLUTIONS		INTEGRATED-SOLUTIONS-68K
INTEGRATED-SOLUTIONS-CREATOR		INTEL-IPSC
INTEGRATED-SOLUTIONS-CREATOR-8		INTEL-386
IS-1	IS-68010	LMI
LSI-11	LSI-11/2	LSI-11/23
LSI-11/73	M68000	MAC-II
MASSCOMP	MC500	MC68000
MICROPORT	MICROVAX	MICROVAX-I
MV/8000	NAS3-5	NCR-COMTEN-3690
NEXT/N1000-316	NOW	ONYX-Z8000
PDP-11	PDP-11/23	PDP-11/24
PDP-11/3	PDP-11/34	PDP-11/40
PDP-11/44	PDP-11/45	PDP-11/50
PDP-11/70	PDP-11/73	PE-3205
PE-7/32	PERQ	PLEXUS-P/60

PLI PLURIBUS PRIME-2250
PRIME-2250 PRIME-2350 PRIME-2450
PRIME-2655 PRIME-2755 PRIME-550II
PRIME-750 PRIME-850 PRIME-9650
PRIME-9655 PRIME-9750 PRIME-9755
PRIME-9950 PRIME-9955 PRIME-9955II
PYRAMID-90 PYRAMID-90MX PYRAMID-90X
RIDGE RIDGE-32 RIDGE-32C
ROLM-1666 S1-MKIIA SEQUENT-BALANCE-8000
SGI-IRIS-2400 SGI-IRIS-2500 SGI-IRIS-3010
SGI-IRIS-3020 SGI-IRIS-3030 SGI-IRIS-3110
SGI-IRIS-3115 SGI-IRIS-3120 SGI-IRIS-3130
SGI-IRIS-4D/120GTX SGI-IRIS-4D/120S SGI-IRIS-4D/20
SGI-IRIS-4D/20G SGI-IRIS-4D/210GTX SGI-IRIS-4D/210S
SGI-IRIS-4D/220GTX SGI-IRIS-4D/220S SGI-IRIS-4D/240GTX
SGI-IRIS-4D/240S SGI-IRIS-4D/25 SGI-IRIS-4D/25G
SGI-IRIS-4D/25S SGI-IRIS-4D/280GTX SGI-IRIS-4D/280S
SGI-IRIS-4D/50 SGI-IRIS-4D/50G SGI-IRIS-4D/50GT
SGI-IRIS-4D/60 SGI-IRIS-4D/60G SGI-IRIS-4D/60GT
SGI-IRIS-4D/60T SGI-IRIS-4D/70 SGI-IRIS-4D/70G
SGI-IRIS-4D/70GT SGI-IRIS-4D/80GT SGI-IRIS-4D/80S
SGI-IRIS-4SERVER-8 SGI-IRIS-CS/12 SIEMENS
SILICON-GRAPHICS SILICON-GRAPHICS-IRIS SMI
SPERRY-DCP/10 SUN SUN-100
SUN-120 SUN-130 SUN-150
SUN-170 SUN-2 SUN-2/100
SUN-2/120 SUN-2/130 SUN-2/140
SUN-2/150 SUN-2/160 SUN-2/170
SUN-2/50 SUN-3/110 SUN-3/140
SUN-3/150 SUN-3/160 SUN-3/180
SUN-3/200 SUN-3/260 SUN-3/280
SUN-3/470 SUN-3/480 SUN-3/50
SUN-3/60 SUN-3/75 SUN-3/80
SUN-386i/250 SUN-4/110 SUN-4/150
SUN-4/200 SUN-4/260 SUN-4/280
SUN-4/330 SUN-4/370 SUN-4/390
SUN-4/60 SUN-50 SUN-68000
SYMBOLICS-3600 SYMBOLICS-3670 SYMMETRIC-375
SYMULT TANDEM-TXP TANDY-6000
TEK-6130 TI-EXPLORER TP-4000
TRS-80 UNIVAC-1100 UNIVAC-1100/60
UNIVAC-1100/62 UNIVAC-1100/63 UNIVAC-1100/64
UNIVAC-1100/70 UNIVAC-1160 UNKNOWN
VAX-11/725 VAX-11/730 VAX-11/750
VAX-11/780 VAX-11/785 VAX-11/790
VAX-11/8600 VAX-8600 WANG-PC002
WANG-VS100 WANG-VS400 WYSE-386
XEROX-1108 XEROX-8010 ZENITH-148

System Names

These are the Official System Names as they appear in the Domain Name System WKS records and the NIC Host Table. Their use is described in RFC-952 [53].

A system name may be up to 40 characters taken from the set of upper- case letters, digits, and the two punctuation characters hyphen and slash. It must start with a letter, and end with a letter or digit.

AEGIS	MACOS	TP3010
APOLLO	MINOS	TRSDOS
BS-2000	MOS	ULTRIX
CEDAR	MPE5	UNIX
CGW	MSDOS	UNIX-BSD
CHORUS	MULTICS	UNIX-V1AT
CHRYSALIS	MVS	UNIX-V
CMOS	MVS/SP	UNIX-V.1
CMS	NEXUS	UNIX-V.2
COS	NMS	UNIX-V.3
CPIX	NONSTOP	UNIX-PC
CTOS	NOS-2	UNKNOWN
CTSS	OS/DDP	UT2D
DCN	OS4	V
DDNOS	OS86	VM
DOMAIN	OSX	VM/370
DOS	PCDOS	VM/CMS
EDX	PERQ/OS	VM/SP
ELF	PLI	VMS
EMBOS	PSDOS/MIT	VMS/EUNICE
EMMOS	PRIMOS	VRTX
EPOS	RMX/RDOS	WAITS
FOONEX	ROS	WANG
FUZZ	RSX11M	X11R3
GCOS	SATOPS	XDE
GPOS	SCO-XENIX/386	XENIX
HDOS	SCS	
IMAGEN	SIMP	
INTERCOM	SUN	
IMPRESS	SUN OS 3.5	
INTERLISP	SUN OS 4.0	
IOS	SWIFT	
IRIX	TAC	
ISI-68020	TANDEM	
ITS	TENEX	
LISP	TOPS10	
LISPM	TOPS20	
LOCUS	TOS	

RFC-1060 Assigned Numbers March 1990

Protocol And Service Names

These are the Official Protocol Names as they appear in the Domain Name System WKS records and the NIC Host Table. Their use is described in RFC-952 [53].

A protocol or service may be up to 40 characters taken from the set of uppercase letters, digits, and the punctuation character hyphen. It must start with a letter, and end with a letter or digit.

ARGUS	- ARGUS Protocol
ARP	- Address Resolution Protocol
AUTH	- Authentication Service
BBN-RCC-MON	- BBN RCC Monitoring
BL-IDM	- Britton Lee Intelligent Database Machine
BOOTP	- Bootstrap Protocol
BOOTPC	- Bootstrap Protocol Client
BOOTPS	- Bootstrap Protocol Server
BR-SAT-MON	- Backroom SATNET Monitoring
CFTP	- CFTP
CHAOS	- CHAOS Protocol
CHARGEN	- Character Generator Protocol
CISCO-FNA	- CISCO FNATIVE
CISCO-TNA	- CISCO TNATIVE
CISCO-SYS	- CISCO SYSMAINT
CLOCK	- DCNET Time Server Protocol
CMOT	- Common Mgmnt Info Services and Protocol over TCP/IP
COOKIE-JAR	- Authentication Scheme
CSNET-NS	- CSNET Mailbox Nameserver Protocol
DAYTIME	- Daytime Protocol
DCN-MEAS	- DCN Measurement Subsystems Protocol
DCP	- Device Control Protocol
DGP	- Dissimilar Gateway Protocol
DISCARD	- Discard Protocol
DOMAIN	- Domain Name System
ECHO	- Echo Protocol
EGP	- Exterior Gateway Protocol
EMCON	- Emission Control Protocol
EMFIS-CNTL	- EMFIS Control Service
EMFIS-DATA	- EMFIS Data Service
FINGER	- Finger Protocol
FTP	- File Transfer Protocol
FTP-DATA	- File Transfer Protocol Data
GGP	- Gateway Gateway Protocol
GRAPHICS	- Graphics Protocol
HMP	- Host Monitoring Protocol
HOST2-NS	- Host2 Name Server
HOSTNAME	- Hostname Protocol
ICMP	- Internet Control Message Protocol
IGMP	- Internet Group Management Protocol
IGP	- Interior Gateway Protocol
IMAP2	- Interim Mail Access Protocol version 2

INGRES-NET - INGRES-NET Service
IP - Internet Protocol
IPCU - Internet Packet Core Utility
IPPC - Internet Pluribus Packet Core
IP-ARC - Internet Protocol on ARCNET
IP-ARPA - Internet Protocol on ARPANET
IP-DC - Internet Protocol on DC Networks
IP-DVMRP - Distance Vector Multicast Routing Protocol
IP-E - Internet Protocol on Ethernet Networks
IP-EE - Internet Protocol on Exp. Ethernet Nets
IP-FDDI - Transmission of IP over FDDI
IP-HC - Internet Protocol on Hyperchannnel
IP-IEEE - Internet Protocol on IEEE 802
IP-IPX - Transmission of 802.2 over IPX Networks
IP-MTU - IP MTU Discovery Options
IP-NETBIOS - Internet Protocol Datagrams over NetBIOS Networks
IP-SLIP - Transmission of IP over Serial Lines
IP-WB - Internet Protocol on Wideband Network
IP-X25 - Internet Protocol on X.25 Networks
IRTP - Internet Reliable Transaction Protocol
ISI-GL - ISI Graphics Language Protocol
ISO-TP4 - ISO Transport Protocol Class 4
ISO-TSAP - ISO TSAP
LA-MAINT - IMP Logical Address Maintenance
LARP - Locus Address Resoultion Protocol
LDP - Loader Debugger Protocol
LEAF-1 - Leaf-1 Protocol
LEAF-2 - Leaf-2 Protocol
LINK - Link Protocol
LOC-SRV - Location Service
LOGIN - Login Host Protocol
MAIL - Format of Electronic Mail Messages
MERIT-INP - MERIT Internodal Protocol
METAGRAM - Metagram Relay
MIB - Management Information Base
MIT-ML-DEV - MIT ML Device
MFE-NSP - MFE Network Services Protocol
MIT-SUBNET - MIT Subnet Support
MIT-DOV - MIT Dover Spooler
MPM - Internet Message Protocol (Multimedia Mail)
MPM-FLAGS - MPM Flags Protocol
MPM-SND - MPM Send Protocol
MSG-AUTH - MSG Authentication Protocol
MSG-ICP - MSG ICP Protocol
MUX - Multiplexing Protocol
NAMESERVER - Host Name Server
NETBIOS-DGM - NETBIOS Datagram Service
NETBIOS-NS - NETBIOS Name Service
NETBIOS-SSN - NETBIOS Session Service
NETBLT - Bulk Data Transfer Protocol
NETED - Network Standard Text Editor
NETRJS - Remote Job Service
NI-FTP - NI File Transfer Protocol
NI-MAIL - NI Mail Protocol
NICNAME - Who Is Protocol

NFILE - A File Access Protocol
NNTP - Network News Transfer Protocol
NSW-FE - NSW User System Front End
NTP - Network Time Protocol
NVP-II - Network Voice Protocol
OSPF - Open Shortest Path First Interior GW Protocol
PCMAIL - Pcmail Transport Protocol
POP2 - Post Office Protocol - Version 2
POP3 - Post Office Protocol - Version 3
PPP - Point-to-Point Protocol
PRM - Packet Radio Measurement
PUP - PUP Protocol
PWDGEN - Password Generator Protocol
QUOTE - Quote of the Day Protocol
RARP - A Reverse Address Resolution Protocol
RATP - Reliable Asynchronous Transfer Protocol
RDP - Reliable Data Protocol
RIP - Routing Information Protocol
RJE - Remote Job Entry
RLP - Resource Location Protocol
RTELNET - Remote Telnet Service
RVD - Remote Virtual Disk Protocol
SAT-EXPAK - Satnet and Backroom EXPAK
SAT-MON - SATNET Monitoring
SEP - Sequential Exchange Protocol
SFTP - Simple File Transfer Protocol
SGMP - Simple Gateway Monitoring Protocol
SNMP - Simple Network Management Protocol
SMI - Structure of Management Information
SMTP - Simple Mail Transfer Protocol
SQLSRV - SQL Service
ST - Stream Protocol
STATSRV - Statistics Service
SU-MIT-TG - SU/MIT Telnet Gateway Protocol
SUN-RPC - SUN Remote Procedure Call
SUPDUP - SUPDUP Protocol
SUR-MEAS - Survey Measurement
SWIFT-RVF - Remote Virtual File Protocol
TACACS-DS - TACACS-Database Service
TACNEWS - TAC News
TCP - Transmission Control Protocol
TELNET - Telnet Protocol
TFTP - Trivial File Transfer Protocol
THINWIRE - Thinwire Protocol
TIME - Time Server Protocol
TP-TCP - ISO Transport Service on top of the TCP
TRUNK-1 - Trunk-1 Protocol
TRUNK-2 - Trunk-2 Protocol
UCL - University College London Protocol
UDP - User Datagram Protocol
NNTP - Network News Transfer Protocol
USERS - Active Users Protocol
UUCP-PATH - UUCP Path Service
VIA-FTP - VIA Systems-File Transfer Protocol
VISA - VISA Protocol

VMTP	- Versatile Message Transaction Protocol
WB-EXPAK	- Wideband EXPAK
WB-MON	- Wideband Monitoring
XNET	- Cross Net Debugger
XNS-IDP	- Xerox NS IDP

Terminal Type Names

These are the Official Terminal Type Names. Their use is described in RFC-930 [128]. The maximum length of a name is 40 characters.

A terminal names may be up to 40 characters taken from the set of upper- case letters, digits, and the two punctuation characters hyphen and slash. It must start with a letter, and end with a letter or digit.

ADDS-CONSUL-980
ADDS-REGENT-100
ADDS-REGENT-20
ADDS-REGENT-200
ADDS-REGENT-25
ADDS-REGENT-40
ADDS-REGENT-60
ADDS-VIEWPOINT
ADDS-VIEWPOINT-60
AED-512
AMPEX-DIALOGUE-210
AMPEX-DIALOGUE-80
AMPEX-210
AMPEX-230
ANDERSON-JACOBSON-510
ANDERSON-JACOBSON-630
ANDERSON-JACOBSON-832
ANDERSON-JACOBSON-841
ANN-ARBOR-AMBASSADOR
ANSI
ARDS
BITGRAPH
BUSSIPLEXER
CALCOMP-565
CDC-456
CDI-1030
CDI-1203
C-ITOH-101
C-ITOH-50
C-ITOH-80
CLNZ
COMPUCOLOR-II
CONCEPT-100
CONCEPT-104
CONCEPT-108
DATA-100
DATA-GENERAL-6053
DATAGRAPHIX-132A
DATAMEDIA-1520
DEC-VT61
DEC-VT62
DELTA-DATA-5000
DELTA-DATA-NIH-7000
DELTA-TELTERM-2
DATAMEDIA-1521
DATAMEDIA-2500
DATAMEDIA-3025
DATAMEDIA-3025A
DATAMEDIA-3045
DATAMEDIA-3045A
DATAMEDIA-DT80/1
DATAPOINT-2200
DATAPOINT-3000
DATAPOINT-3300
DATAPOINT-3360
DEC-DECWRITER-I
DEC-DECWRITER-II
DEC-GIGI
DEC-GT40
DEC-GT40A
DEC-GT42
DEC-LA120
DEC-LA30
DEC-LA36
DEC-LA38
DEC-VT05
DEC-VT100
DEC-VT101
DEC-VT102
DEC-VT125
DEC-VT131
DEC-VT132
DEC-VT200
DEC-VT220
DEC-VT240
DEC-VT241
DEC-VT300
DEC-VT320
DEC-VT340
DEC-VT50
DEC-VT50H
DEC-VT52
DEC-VT55
HP-2649A
IBM-1050
IBM-2741
IBM-3101
IBM-3101-10

DIABLO-1620
DIABLO-1640
DIGILOG-333
DTC-300S
DTC-382
EDT-1200
EXECUPORT-4000
EXECUPORT-4080
FACIT-TWIST-4440
FREEDOM-100
FREEDOM-110
FREEDOM-200
GENERAL-TERMINAL-100A
GENERAL-TERMINAL-101
GIPSI-TX-M
GIPSI-TX-ME
GIPSI-TX-C4
GIPSI-TX-C8
GSI
HAZELTINE-1420
HAZELTINE-1500
HAZELTINE-1510
HAZELTINE-1520
HAZELTINE-1552
HAZELTINE-2000
HAZELTINE-ESPRIT
HP-2392
HP-2621
HP-2621A
HP-2621P
HP-2623
HP-2626
HP-2626A
HP-2626P
HP-2627
HP-2640
HP-2640A
HP-2640B
HP-2645
HP-2645A
HP-2648
HP-2648A
HP-2649
MICROBEE
MICROTERM-ACT-IV
MICROTERM-ACT-V
MICROTERM-ERGO-301
MICROTERM-MIME-1
MICROTERM-MIME-2
MICROTERM-ACT-5A
MICROTERM-TWIST
NEC-5520
NETRONICS
NETWORK-VIRTUAL-TERMINAL
OMRON-8025AG
IBM-3151
IBM-3275-2
IBM-3276-2
IBM-3276-3
IBM-3276-4
IBM-3277-2
IBM-3278-2
IBM-3278-3
IBM-3278-4
IBM-3278-5
IBM-3279-2
IBM-3279-3
IBM-5151
IBM-5154
IBM-5081
IBM-6153
IBM-6154
IBM-6155
IBM-AED
IBM-3278-2-E
IBM-3278-3-E
IBM-3278-4-E
IBM-3278-5-E
IBM-3279-2-E
IBM-3279-3-E
IMLAC
INFOTON-100
INFOTON-400
INFOTONKAS
ISC-8001
LSI-ADM-1
LSI-ADM-11
LSI-ADM-12
LSI-ADM-2
LSI-ADM-20
LSI-ADM-22
LSI-ADM-220
LSI-ADM-3
LSI-ADM-31
LSI-ADM-3A
LSI-ADM-42
LSI-ADM-5
MEMOREX-1240
TELETEC-DATASCREEN
TELETERM-1030
TELETYPE-33
TELETYPE-35
TELETYPE-37
TELETYPE-38
TELETYPE-40
TELETYPE-43
TELEVIDEO-910
TELEVIDEO-912
TELEVIDEO-920
TELEVIDEO-920B

PERKIN-ELMER-550
PERKIN-ELMER-1100
PERKIN-ELMER-1200
PERQ
PLASMA-PANEL
QUME-SPRINT-5
QUME-101
QUME-102
SOROC
SOROC-120
SOUTHWEST-TECHNICAL-PRODUCTS-CT82
SUN
SUPERBEE
SUPERBEE-III-M
TEC
TEKTRONIX-4006
TEKTRONIX-4010
TEKTRONIX-4012
TEKTRONIX-4013
TEKTRONIX-4014
TEKTRONIX-4023
TEKTRONIX-4024
TEKTRONIX-4025
TEKTRONIX-4027
TEKTRONIX-4105
TEKTRONIX-4107
TEKTRONIX-4110
TEKTRONIX-4112
TEKTRONIX-4113
TEKTRONIX-4114
TEKTRONIX-4115
TEKTRONIX-4125
TEKTRONIX-4404
TELERAY-1061
TELERAY-3700
TELERAY-3800
TELEVIDEO-920C
TELEVIDEO-925
TELEVIDEO-955
TELEVIDEO-950
TELEVIDEO-970
TELEVIDEO-975
TERMINET-1200
TERMINET-300
TI-700
TI-733
TI-735
TI-743
TI-745
TI-800
TYCOM
UNIVAC-DCT-500
VIDEO-SYSTEMS-1200
VIDEO-SYSTEMS-5000
VOLKER-CRAIG-303
VOLKER-CRAIG-303A
VOLKER-CRAIG-404
VISUAL-200
VISUAL-55
WYSE-30
WYSE-50
WYSE-60
WYSE-75
WYSE-85
XEROX-1720
XTERM
ZENITH-H19
ZENITH-Z29
ZENTEC-30

[2] BBN, "Specifications for the Interconnection of a Host and an IMP", Report 1822, Bolt Beranek and Newman, Cambridge, Massachusetts, revised, December 1981.

[3] BBN, "User Manual for TAC User Database Tool", Bolt Beranek and Newman, September 1984.

[4] Ben-Artzi, Amatzia, "Network Management for TCP/IP Network: An Overview", 3Com, May 1988.

[5] Bennett, C., "A Simple NIFTP-Based Mail System", IEN 169, University College, London, January 1981.

[6] Bhushan, A., "A Report on the Survey Project", RFC-530, NIC 17375, June 1973.

[7] Bisbey, R., D. Hollingworth, and B. Britt, "Graphics Language (version 2.1)", ISI/TM-80-18, Information Sciences Institute, July 1980.

[8] Boggs, D., J. Shoch, E. Taft, and R. Metcalfe, "PUP: An Internetwork Architecture", XEROX Palo Alto Research Center, CSL-79-10, July 1979; also in IEEE Transactions on Communication, Volume COM-28, Number 4, April 1980.

[12] Bressler, B., "Remote Job Entry Protocol", RFC-407, NIC 12112, October 1972.

[13] Bressler, R., "Inter-Entity Communication -- An Experiment", RFC-441, NIC 13773, January 1973.

[18] Cisco Systems, "Gateway Server Reference Manual", Manual Revision B, January 10, 1988.

[21] Cohen, D., "On Holy Wars and a Plea for Peace", IEEE Computer Magazine, October 1981.

[23] Cohen, D. and J. Postel, "Multiplexing Protocol", IEN 90, Information Sciences Institute, May 1979.

[24] COMPASS, "Semi-Annual Technical Report", CADD-7603-0411, Massachusetts Computer Associates, 4 March 1976. Also as, "National Software Works, Status ReportNo. 1," RADC-TR-76-276, Volume 1, September 1976. And COMPASS. "Second Semi-Annual Report," CADD-7608-1611, Massachusetts Computer Associates, August 1976.

[39] DCA, "3270 Display System Protocol", #1981-08.

[40] DDN Protocol Handbook, "Telnet Output Line Width Option", NIC 50005, December 1985.

[41] DDN Protocol Handbook, "Telnet Output Page Size Option", NIC 50005, December 1985.

[42] DDN Protocol Handbook, "Telnet Reconnection Option", NIC 50005, December 1985.

[45] Feinler, E., editor, "DDN Protocol Handbook", Network Information Center, SRI International, December 1985.

[46] Feinler, E., editor, "Internet Protocol Transition Workbook", Network Information Center, SRI International, March 1982.

[47] Feinler, E. and J. Postel, eds., "ARPANET Protocol Handbook", NIC 7104, for the Defense Communications Agency by SRI International, Menlo Park, California, Revised January 1978.

[49] Forgie, J., "ST - A Proposed Internet Stream Protocol", IEN 119, MIT Lincoln Laboratory, September 1979.

[50] Forsdick, H., "CFTP", Network Message, Bolt Beranek and Newman, January 1982.

[56] Haverty, J., "XNET Formats for Internet Protocol Version 4", IEN 158, October 1980.

[66] Kent, S., and J. Linn, "Privacy Enhancement for Internet Electronic Mail: Part II -- Certificate-Based Key Management", BBNCC and DEC, August 1989.

[74] M/A-COM Government Systems, "Dissimilar Gateway Protocol Specification, Draft Version", Contract no. CS901145, November 16, 1987.

[76] Malis, A., "Logical Addressing Implementation Specification", BBN Report 5256, pp 31-36, May 1983.

[77] Malkin, G., "KNET/VM Command Message Protocol Functional Overview", Spartacus, Inc., January 4, 1988.

[78] Metcalfe, R. M. and D. R. Boggs, "Ethernet: Distributed Packet Switching for Local Computer Networks", Communications of the ACM, 19 (7), pp 395-402, July1976.

[85] NSW Protocol Committee, "MSG: The Interprocess Communication Facility for the National Software Works", CADD-7612-2411, Massachusetts Computer Associates,BBN 3237, Bolt Beranek and Newman, Revised December 1976.

[99] Postel, J., "Name Server", IEN 116, Information Sciences Institute, August 1979.

[124] Shuttleworth, B., "A Documentary of MFENet, a National Computer Network", UCRL-52317, Lawrence Livermore Labs, Livermore, California, June 1977.

[127] Solomon, M., L. Landweber, and D. Neuhengen, "The CSNET Name Server", Computer Networks, v.6, n.3, pp. 161-172, July 1982.

[129] Sproull, R., and E. Thomas, "A Networks Graphics Protocol", NIC 24308, August 1974.

[132] Taylor, J., "ERPC Functional Specification", Version 1.04, HYDRA Computer Systems, Inc., July 1984.

[133] "The Ethernet, A Local Area Network: Data Link Layer and Physical Layer Specification", AA-K759B-TK, Digital Equipment Corporation, Maynard, MA. Also as: "The Ethernet - A Local Area Network", Version 1.0, Digital Equipment Corporation, Intel Corporation, Xerox Corporation, September 1980. And: "The Ethernet,A Local Area Network: Data Link Layer and Physical Layer Specifications", Digital, Intel and Xerox, November 1982. And: XEROX, "The Ethernet, A Local Area Network: Data Link Layer and Physical Layer Specification", X3T51/80-50, Xerox Corporation, Stamford, CT., October 1980.

[134] The High Level Protocol Group, "A Network Independent File Transfer Protocol", INWG Protocol Note 86, December 1977.

[135] Thomas, Bob, "The Interhost Protocol to Support CRONUS/DIAMOND Interprocess Communication", BBN, September 1983.

[137] Uttal, J., J. Rothschild, and C. Kline, "Transparent Integration of UNIX and MS-DOS", Locus Computing Corporation.

[143] Welch, B., "The Sprite Remote Procedure Call System", Technical Report, UCB/Computer Science Dept., 86/302, University of California at Berkeley, June 1986.

[144] Xerox, "Courier: The Remote Procedure Protocol", XSIS 038112, December 1981.

[AB20] Art Berggreen ACC art@SALT.ACC.ARPA

[ABB2] A. Blasco Bonito CNUCE blasco@ICNUCEVM.CNUCE.CNR.IT

[AD14] Annette DeSchon ISI DESCHON@ISI.EDU

[AGM] Andy Malis BBN Malis@BBN.COM

[AKH5] Arthur Hartwig UQNET munnari!wombat.decnet.uq.oz.au!
ccarthur@UUNET.UU.NET

[ANM2] April N. Marine SRI APRIL@NIC.DDN.MIL

[AW90] Amanda Walker Intercon AMANDA@INTERCON.COM

[AXB] Albert G. Broscius UPENN broscius@DSL.CIS.UPENN.EDU

[AXB1] Amatzia Ben-Artzi ---none---

[AXC] Andrew Cherenson SGI arc@SGI.COM

[AXC1] Anthony Chung Sytek sytek!syteka!anthony@HPLABS.HP.COM

[AXC2] Asheem Chandna AT&T ac0@mtuxo.att.com

[AXM] Alex Martin Retix ---none---

[AXS] Arthur Salazar Locus lcc.arthur@SEAS.UCLA.EDU

[BA4] Brian Anderson BBN baanders@CCQ.BBN.COM

[BB257] Brian W. Brown SynOptics BBROWN@MVIS1.SYNOPTICS.COM

[BCH2] Barry Howard LLL Howard@NMFECC.ARPA

[BCN] Clifford B. Newman UWASH bcn@CS.WASHINGTON.EDU

[BD70] Bernd Doleschal SEL Doleschal@A.ISI.EDU

[BH144] Bridget Halsey Banyan bah@BANYAN.BANYAN.COM

[BJR2] Bill Russell NYU russell@cmcl2.NYU.EDU

[BKR] Brian Reid DEC reid@DECWRL.DEC.COM

[BP52] Brad Parker CAYMAN brad@cayman.Cayman.COM

[BS221] Bob Stewart Xyplex STEWART@XYPLEX.COM

[BWB6] Barry Boehm DARPA boehm@DARPA.MIL

[BXA] Bill Anderson MITRE wda@MITRE-BEDFORD.ORG

[BXB] Brad Benson Touch ---none---

[BXE] Brian A. Ehrmantraut Auspex Systems bae@auspex.com

[BXH] Brian Horn Locus ---none---

[BXL] Brian Lloyd SIRIUS ---none---

[BXN] Bill Norton Merit wbn@MERIT.EDU

[BXV] Bill Versteeg NRC bvs@NRC.COM

[BXW] Brent Welch Sprite brent%sprite.berkeley.edu@GINGER.BERKELEY.EDU

[BXW1] Bruce Willins Raycom ---none---

[BXZ] Bob Zaniolo Reuter ---none---

[CLH3] Charles Hedrick RUTGERS HEDRICK@ARAMIS.RUTGERS.EDU

[CMR] Craig Rogers ISI Rogers@ISI.EDU

[CXM] Charles Marker II MIPS marker@MIPS.COM

[CXT] Christopher Tengi Princeton tengi@Princeton.EDU

[DAG4] David A. Gomberg MITRE gomberg@GATEWAY.MITRE.ORG

[DB14] Dave Borman Cray dab@CRAY.COM

[DC126] Dick Cogger Cornell rhx@CORNELLC.CIT.CORNELL.EDU

[DCP1] David Plummer MIT DCP@SCRC-QUABBIN.ARPA

[DDC1] David Clark MIT ddc@LCS.MIT.EDU

[DJK13] David Kaufman DeskTalk ---none---

[DLM1] David Mills LINKABIT Mills@HUEY.UDEL.EDU

[DM28] Dennis Morris DCA Morrisd@IMO-UVAX.DCA.MIL

[DM280] Dave Mackie NCD lupine!djm@UUNET.UU.NET

[DM354] Don McWilliam UBC mcwillm@CC.UBC.CA

[DPR] David Reed MIT-LCS Reed@MIT-MULTICS.ARPA

[DRC3] Dave Cheriton STANFORD cheriton@PESCADERO.STANFORD.EDU

[DT15] Daniel Tappan BBN Tappan@BBN.COM

[DW181] David Wolfe SRI ctabka@TSCA.ISTC.SRI.COM

[DW183] David Waitzman BBN dwaitzman@BBN.COM

[DXB] Dave Buehmann Intergraph ingr!daveb@UUNET.UU.NET

[DXD] Dennis J.W. Dube VIA SYSTEMS ---none---

[DXG] David Goldberg SMI sun!dg@UCBARPA.BERKELEY.EDU

[DXK] Doug Karl OSU KARL-D@OSU-20.IRCC.OHIO-STATE.EDU

[DXM] Didier Moretti Ungermann-Bass ---none---

[DXM1] Donna McMalster David Systems ---none---

[DXP] Dave Preston CMC ---none---

[DY26] Dennis Yaro SUN yaro@SUN.COM

[EAK4] Earl Killian LLL EAK@MORDOR.S1.GOV

[EBM] Eliot Moss MIT EBM@XX.LCS.MIT.EDU

[EP53] Eric Peterson Locus lcc.eric@SEAS.UCLA.EDU

[EXC] Ed Cain DCA cain@edn-unix.dca.mil

[EXR] Eric Rubin FiberCom err@FIBERCOM.COM

[EXR1] Efrat Ramati Lannet Co. ---none---

[FB77] Fred Baker Vitalink baker%vitam6@UUNET.UU.NET

[FJK2] Frank Kastenholz Interlan KASTEN@MITVMA.MIT.EDU

[FJW] Frank J. Wancho WSMR WANCHO@SIMTEL20.ARPA

[FXB1] Felix Burton DIAB FB@DIAB.SE

[GAL5] Guillermo A. Loyola IBM LOYOLA@IBM.COM

[GB7] Gerd Beling FGAN GBELING@ISI.EDU

[GEOF] Geoff Goodfellow OSD Geoff@FERNWOOD.MPK.CA.US

[GGB2] Geoff Baehr SUN geoffb@ENG.SUN.COM

[GM23] Glenn Marcy CMU Glenn.Marcy@A.CS.CMU.EDU

[GS2] Greg Satz cisco satz@CISCO.COM

[GS123] Geof Stone NSC geof@NETWORK.COM

[GSM11] Gary S. Malkin Proteon gmalkin@PROTEON.COM

[GXG] Gil Greebaum Unisys gcole@nisd.cam.unisys.com

[GXP] Gill Pratt MIT gill%mit-ccc@MC.LCS.MIT.EDU

[GXS] Guenther Schreiner LINK guenther%ira.uka.de@RELAY.CS.NET

[GXT] Glenn Trewitt STANFORD trewitt@AMADEUS.STANFORD.EDU

[GXT1] Gene Tsudik USC tsudik@USC.EDU

[GXW] Glenn Waters Bell Northern gwaters@BNR.CA

[HCF2] Harry Forsdick BBN Forsdick@BBN.COM

[HS23] Hokey Stenn Plus5 hokey@PLUS5.COM

[HWB] Hans-Werner Braun MICHIGAN HWB@MCR.UMICH.EDU

[HXE] Hunaid Engineer Cray hunaid@OPUS.CRAY.COM

[HXK] Henry Kaijak Gandalf ---none---

[IEEE] Vince Condello IEEE ---none---

[JAG] James Gosling SUN JAG@SUN.COM

[JB478] Jonathan Biggar Netlabs jon@netlabs.com

[JBP] Jon Postel ISI Postel@ISI.EDU

[JBW1] Joseph Walters, Jr. BBN JWalters@BBN.COM

[JCB1] John Burruss BBN JBurruss@VAX.BBN.COM

[JCM48] Jeff Mogul DEC mogul@DECWRL.DEC.COM

[JD21] Jonathan Dreyer BBN Dreyer@CCV.BBN.COM

[JDC20] Jeffrey Case UTK case@UTKUX1.UTK.EDU

[JFH2] Jack Haverty BBN JHaverty@BBN.COM

[JFW] Jon F. Wilkes STC Wilkes@CCINT1.RSRE.MOD.UK

[JGH] Jim Herman BBN Herman@CCJ.BBN.COM

[JJB25] John Bowe BBN jbowe@PINEAPPLE.BBN.COM

[JKR1] Joyce K. Reynolds ISI JKRey@ISI.EDU

[JR35] Jon Rochlis MIT jon@ATHENA.MIT.EDU

[JRL3] John LoVerso Xylogics loverso@XYLOGICS.COM

[JS28] John A. Shriver Proteon jas@PROTEON.COM

[JTM4] John Moy Proteon jmoy@PROTEON.COM

[JWF] Jim Forgie MIT/LL FORGIE@XN.LL.MIT.EDU

[JXB] Jeffrey Buffun Apollo jbuffum@APOLLO.COM

[JXC] John Cook Chipcom cook@chipcom.com

[JXE2] Jeanne Evans UKMOD JME%RSRE.MOD.UK@CS.UCL.AC.UK

[JXF] Josh Fielk Optical Data Systems ---none---

[JXG] Jerry Geisler Boeing ---none---

[JXG1] Jim Greuel HP jimg%hpcndpc@hplabs.hp.com

[JXH] Jeff Honig Cornell jch@sonne.tn.cornell.edu

[JXH1] Jim Hayes Apple Hayes@APPLE.COM

[JXI] Jon Infante ICL ---none---

[JXM] Joseph Murdock Network Resources Corporation ---none---

[JXO] Jack O'Neil ENCORE ---none---

[JXO1] Jerrilynn Okamura Ontologic ---none---

[JXO2] Jarkko Oikarinen Tolsun jto@TOLSUN.OULU.FI

[JXP] Joe Pato Apollo apollo!pato@EDDIE.MIT.EDU

[JXR] Jacob Rekhter IBM Yakov@IBM.COM

[JXS] Jim Stevens Rockwell Stevens@ISI.EDU

[JXS1] John Sancho CastleRock ---none---

[KAA] Ken Adelman TGV, Inc. Adelman@TGV.COM

[KA4] Karl Auerbach Epilogue auerbach@csl.sri.com

[KH43] Kathy Huber BBN khuber@bbn.com

[KLH] Ken Harrenstien SRI KLH@NIC.DDN.MIL

[KR35] Keith Reynolds SCO keithr@SCO.COM

[KSL] Kirk Lougheed cisco LOUGHEED@MATHOM.CISCO.COM

[KXD] Kevin DeVault NI ---none---

[KXS] Keith Sklower Berkeley sklower@okeeffe.berkeley.edu

[KXW] Ken Whitfield MCNC ken@MCNC.ORG

[KZM] Keith McCloghrie TWG kzm@TWG.ARPA

[LL69] Lawrence Lebahn DIA DIA3@PAXRV-NES.NAVY.MIL

[LLP] Larry Peterson ARIZONA llp@ARIZONA.EDU

[LXE] Len Edmondson SUN len@TOPS.SUN.COM

[LXF] Larry Fischer DSS lfischer@dss.com

[LXH] Leo Hourvitz NeXt leo@NEXT.COM

[MA] Mike Accetta CMU MIKE.ACCETTA@CMU-CS-A.EDU

[MARY] Mary K. Stahl SRI Stahl@NIC.DDN.MIL

[MAR10] Mark A. Rosenstein MIT mar@ATHENA.MIT.EDU

[MB] Michael Brescia BBN Brescia@CCV.BBN.COM

[MBG] Michael Greenwald SYMBOLICS Greenwald@SCRC-STONY-BROOK.ARPA

[MCSJ] Mike StJohns TPSC StJohns@MIT-MULTICS.ARPA

[ME38] Marc A. Elvy Marble ELVY@CARRARA.MARBLE.COM

[MKL] Mark Lottor SRI MKL@NIC.DDN.MIL

[ML109] Mike Little MACOM little@MACOM4.ARPA

[MLS34] L. Michael Sabo TMAC darth!eniac!sabo@Sun.Com

[MO2] Michael O'Brien AEROSPACE obrien@AEROSPACE.AERO.ORG

[MRC] Mark Crispin Simtel MRC@SIMTEL20.ARPA

[MS9] Marty Schoffstahl Nysernet schoff@NISC.NYSER.NET

[MS56] Marvin Solomon WISC solomon@CS.WISC.EDU

[MXB] Mike Berrow Relational Technology ---none---

[MXB1] Mike Burrows DEC burrows@SRC.DEC.COM

[MXL] Mark L. Lambert MIT markl@PTT.LCS.MIT.EDU

[MXP] Martin Picard Oracle ---none---

[MXS] Mike Spina Prime WIZARD%enr.prime.com@RELAY.CS.NET

[MXW] Michael Waters EON ---none---

[NC3] J. Noel Chiappa MIT JNC@XX.LCS.MIT.EDU

[NT12] Neil Todd IST mcvax!ist.co.uk!neil@UUNET.UU.NET

[PAM6] Paul McNabb RICE pam@PURDUE.EDU

[PCW] C. Philip Wood LANL cpw@LANL.GOV

[PD39] Pete Delaney ECRC pete%ecrcvax@CSNET-RELAY.ARPA

[PHD1] Pieter Ditmars BBN pditmars@BBN.COM

[PK] Peter Kirstein UCL Kirstein@NSS.CS.UCL.AC.UK

[PL4] Phil Lapsley BERKELEY phil@UCBARPA.BERKELEY.EDU

[PM1] Paul Mockapetris ISI PVM@ISI.EDU

[PXK] Philip Koch Dartmouth Philip.Koch@DARTMOUTH.EDU

[RAM57] Rex Mann CDC ---none---

[RDXS] R. Dwight Schettler HP rds%hpcndm@HPLABS.HP.COM

[RH6] Robert Hinden BBN Hinden@CCV.BBN.COM

[RHT] Robert Thomas BBN BThomas@F.BBN.COM

[RN6] Rudy Nedved CMU Rudy.Nedved@CMU-CS-A.EDU

[RTB3] Bob Braden ISI Braden@ISI.EDU

[RWS4] Robert W. Scheifler ARGUS RWS@XX.LCS.MIT.EDU

[RXB] Ramesh Babu Excelan mtxinu!excelan!ramesh@UCBVAX.BERKELEY.EDU

[RXB1] Ron Bhanukitsiri DEC rbhank@DECVAX.DEC.COM

[RXC] Rob Chandhok CMU chandhok@gnome.cs.cmu.edu

[RXC1] Rick Carlos TI rick.ticipa.csc.ti.com

[RXD] Roger Dev Cabletron ---none---

[RXD1] Ralph Droms NRI rdroms@NRI.RESTON.VA.US

[RXH] Reijane Huai Cheyenne sibal@CSD2.NYU.EDU

[RXJ] Ronald Jacoby SGI rj@SGI.COM

[RXM] Robert Myhill BBN Myhill@CCS.BBN.COM

[RXN] Rina Nethaniel RND ---none---

[RXS] Ron Strich SSDS ---none---

[RXT] Ron Thornton GenRad thornton@qm7501.genrad.com

[RXZ] Rayan Zachariassen Toronto rayan@AI.TORONTO.EDU

[SA1] Sten Andler IBM
andler.ibm-sj@RAND-RELAY.ARPA

[SAF3] Stuart A. Friedberg UWISC stuart@CS.WISC.EDU

[SB98] Stan Barber BCM SOB@BCM.TMC.EDU

[SC3] Steve Casner ISI Casner@ISI.EDU

[SGC] Steve Chipman BBN Chipman@F.BBN.COM

[SHB] Steven Blumenthal BBN BLUMENTHAL@VAX.BBN.COM

[SH37] Sergio Heker JVNC heker@JVNCC.CSC.ORG

[SL70] Stuart Levy UMN slevy@UC.MSC.UMN.EDU

[SRN1] Stephen Northcutt NSWC SNORTHC@RELAY-NSWC.NAVY.MIL

[SS92] Steve Schoch NASA SCHOCH@AMES.ARC.NASA.GOV

[SXA] Susie Armstrong XEROX Armstrong.wbst128@XEROX.COM

[SXB] Scott Bellows Purdue smb@cs.purdue.edu

[SXC] Steve Conklin Intergraph tesla!steve@ingr.com

[SXD] Steve Deering Stanford deering@PECASERO.STANFORD.EDU

[SXH] Steven Hunter LLNL hunter@CCC.MFECC.LLNL.GOV

[SXK] Skip Koppenhaver DAC stubby!skip@uunet.UU.NET

[SXL] Sam Lau Pirelli/Focom ---none---

[SXP] Sanand Patel Canstar sanand@HUB.TORONTO.EDU

[SXS] Steve Silverman MITRE Blankert@MITRE-GATEWAY.ORG

[SXS1] Susie Snitzer Britton-Lee ---none---

[SXW] Steve Waldbusser CMU sw01+@andrew.cmu.edu

[TB6] Todd Baker 3COM tzb@BRIDGE2.3COM.COM

[TC27] Thomas Calderwood BBN TCALDERW@BBN.COM

[TN] Thomas Narten Purdue narten@PURDUE.EDU

[TU] Tom Unger UMich tom@CITI.UMICH.EDU

[TXM] Trudy Miller ACC Trudy@ACC.ARPA

[TXR] Tim Rylance Praxis praxis!tkr@UUNET.UU.NET

[TXS] Ted J. Socolofsky Spider Teds@SPIDER.CO.UK

[UB3] Ulf Bilting CHALMERS bilting@PURDUE.EDU

[UW2] Unni Warrier Netlabs unni@NETLABS.COM

[VXS] Vinod Singh Unify ---none---

[VXT] V. Taylor CANADA vktaylor@NCS.DND.CA

[WDW11] William D. Wisner wisner@HAYES.FAI.ALASKA.EDU

[WJC2] Bill Croft STANFORD Croft@SUMEX-AIM.STANFORD.EDU

[WJS1] Weldon J. Showalter DCA Gamma@EDN-UNIX.ARPA

[WLB8] William L. Biagi Advintech
CSS002.BLBIAGI@ADVINTECH-MVS.ARPA

[WM3] William Melohn SUN Melohn@SUN.COM

[WXS] Wayne Schroeder SDSC schroeder@SDS.SDSC.EDU

[VXW] Val Wilson Spider cvax!spider.co.uk!val@uunet.UU.NET

[YXK] Yoav Kluger Spartacus ykluger@HAWK.ULOWELL.EDU

[YXW] Y.C. Wang Network Application Technology ---none---

[XEROX] Fonda Pallone Xerox ---none---

[ZSU] Zaw-Sing Su SRI ZSu@TSCA.ISTC.SRI.COM

RFC-1073 Telnet Window Size Option

D. Waitzman; BBN STC

October 1988

This RFC describes a proposed Telnet option to allow a client to convey window size to a Telnet server. Distribution of this memo is unlimited.

Command Name and Option Code

NAWS (Negotiate About Window Size) 31

Default Specification

WON'T NAWS
DON'T NAWS

This option does not assume any default window size information. Often the terminal type, passed with the TERMINAL TYPE Telnet option, may imply a window size, but that is not necessary for this option.

RFC-1073 Telnet Window Size Option

Command Meanings

IAC WILL NAWS

Sent by the Telnet client to suggest that NAWS be used.

IAC WON'T NAWS

Sent by the Telnet client to refuse to use NAWS.

IAC DO NAWS

Sent by the Telnet server to suggest that NAWS be used.

IAC DON'T NAWS

Sent by the Telnet server to refuse to use NAWS.

IAC SB NAWS <16-bit value> <16-bit value> IAC SE

Sent by the Telnet client to inform the Telnet server of the window width and height.

The window size information is conveyed via this option from the Telnet client to the Telnet server. The information is advisory. The server may accept the option, but not use the information that is sent.

The client and server negotiate sending the window size information using the standard Telnet WILL/DO/DON'T/WON'T mechanism. If the client and server agree, the client may then send a subnegotiation to convey the window size. If the client's window size is later changed (for instance, the window size is altered by the user), the client may again send the subnegotiation. Because certain operating systems, on which a server may be executing, may not allow the window size information to be updated, the server may send a DON'T NAWS to the client to forbid further subnegotiation after it was initially accepted. A negotiation loop will not form following these rules.

The subnegotiation consists of two values, the width and the height of the window in characters. The values are each sent as two bytes, in the Internet standard byte and bit order. This allows a maximum window width or height of 65535 characters. A value equal to zero is acceptable for the width (or height), and means that no character width (or height) is being sent. In this case, the width (or height) that will be assumed by the Telnet server is operating system specific (it will probably be based upon the terminal type information that may have been sent using the TERMINAL TYPE Telnet option).

The syntax for the subnegotiation is:

IAC SB NAWS WIDTH[1] WIDTH[0] HEIGHT[1] HEIGHT[0] IAC SE

As required by the Telnet protocol, any occurrence of 255 in the subnegotiation must be doubled to distinguish it from the IAC character (which has a value of 255).

RFC-1073 Telnet Window Size Option

Motivation

With the increasing popularity of windowing systems, a Telnet client is often run inside a variable-sized window, and the Telnet server needs to know the window size for proper cursor control. The window may also have its size changed during the Telnet session and the updated window size needs to be conveyed to the server. This memo specifies an option to send the window height and width in characters from a client to a server.

The Telnet options Negotiate Output Line Width (NAOL) and Negotiate Output Page Size (NAOP) do not have the correct semantics for this purpose, and they are not in common use [see RFC-1011 "Official Internet Protocols", and the "Defense Protocol Handbook"]. The NAOL and NAOP options are bidirectional (i.e., the server might control the client's line width or page size), and are limited to 253 characters in each axis.

This option is a better model of the normal window negotiation process. The client has total control over the size of its window and simply tells the server what the current window size is. Furthermore, the 253 character height and width limitation is too low so the new option has a limit of 65535 characters. Finally, this option sends the window height and width concurrently because they are typically changed simultaneously and many operating systems and windowing applications prefer to think in terms of simultaneous changes in height and width.

RFC-1073 Telnet Window Size Option

Description and Implementation Notes

A typical user of this option might be a Telnet client running under X. After a user resizes the client's window, this must be communicated to the Telnet client. In 4.3 BSD Unix, the signal SIGWINCH (window changed) might be caught by the Telnet process and a new NAWS subnegotiation sent to the server. Upon receipt of a NAWS subnegotiation, the server might do the appropriate ioctl to handle the new information, and then could send a SIGWINCH to its child, probably a shell.

Examples

In the following examples all numbers in the data stream are in decimal.

1. Server suggest and client agrees to use NAWS.

   (server sends) IAC DO NAWS
   (client sends) IAC WILL NAWS
   (client sends) IAC SB NAWS 0 80 0 24 IAC SE

   [A window 80 characters wide, 24 characters high]
   [some time occurs and the user changes the window size]

   (client sends) IAC SB NAWS 0 80 0 64 IAC SE

   [A window 80 characters wide, 64 characters high]

   In all numeric form:

   (server sends) 255 253 31
   (client sends) 255 251 31
   (client sends) 255 250 31 0 80 0 24 255 240

   (client sends) 255 250 31 0 80 0 64 255 240

2. Client suggests and server agrees to used NAWS.

   (client sends) IAC WILL NAWS
   (server sends) IAC DO NAWS
   (client sends) IAC SB NAWS 1 44 0 24 IAC SE

   [A window 300 characters wide, 24 characters high]

3. Client suggest and server refuses to use NAWS.

   (client sends) IAC WILL NAWS
   (server sends) IAC DON'T NAWS

4. Server suggests and client refuses to use NAWS.

   (server sends) IAC DO NAWS
   (client sends) IAC WON'T NAWS

Acknowledgments

A more elaborate, X window system specific, version of this option has been implemented at Carnegie-Mellon University by Glenn Marcy and the author. It is widely used in the Carnegie-Mellon University Computer Science Department. Mr. Marcy helped write an early draft of this memo documenting the more elaborate option.

RFC-1079 Telnet Terminal Speed Option

Charles Hedrick; Rutgers University
December 1988

This RFC specifies a standard for the Internet community. Hosts on the Internet that exchange terminal speed information within the Telnet protocol are expected to adopt and implement this standard. Distribution of this memo is unlimited.

This standard is modelled on the "Telnet Terminal Type Option" [RFC-930]. Much of the text of this document is copied from that RFC.

Command Name and Code

TERMINAL-SPEED 32

Default

WON'T TERMINAL-SPEED
DON'T TERMINAL-SPEED

Terminal speed information will not be exchanged.

Command Meanings

IAC WILL TERMINAL-SPEED

Sender is willing to send terminal speed information in a subsequent sub-negotiation.

IAC WON'T TERMINAL-SPEED

Sender refuses to send terminal speed information.

IAC DO TERMINAL-SPEED

Sender is willing to receive terminal speed information in a subsequent sub-negotiation.

IAC DON'T TERMINAL-SPEED

Sender refuses to accept terminal speed information.

IAC SB TERMINAL-SPEED SEND IAC SE

Sender requests receiver to transmit his (the receiver's) terminal speed. The code for SEND is 1. (See below.)

IAC SB TERMINAL-SPEED IS ... IAC SE

Sender is stating his terminal speed. The code for IS is 0. (See below.)

RFC-1079 Telnet Terminal Speed Option

Motivation

Most operating systems have provisions to keep track of the speed (bit rate) of directly attached terminals and modems. This information is used to control various timing-dependent display processes, e.g., the number of padding characters used for delay. Some software also has user interfaces that are tuned differently for fast and slow terminals. The purpose of this option is to provide similar information for telnet connections.

RFC-1079 Telnet Terminal Speed Option

Description of the Option

WILL and DO are used only to obtain and grant permission for future discussion. The actual exchange of status information occurs within option subcommands (IAC SB TERMINAL-SPEED...).

Once the two hosts have exchanged a WILL and a DO, the sender of the DO TERMINAL-SPEED is free to request speed information. Only the sender of the DO may send requests (IAC SB TERMINAL-SPEED SEND IAC SE) and only the sender of the WILL may transmit actual speed information (within an IAC SB TERMINAL-SPEED IS ... IAC SE command). Terminal speed information may not be sent spontaneously, but only in response to a request.

The terminal speed information is an NVT ASCII string. This string contains the decimal representation of the transmit and receive speeds of the terminal, separated by a comma, e.g.,

9600,100

No leading zeros may be included. No extraneous characters such as spaces may be included.

The following is an example of use of the option:

Host1: IAC DO TERMINAL-SPEED
Host2: IAC WILL TERMINAL-SPEED

(Host1 is now free to request status information at any time.)

Host1: IAC SB TERMINAL-SPEED SEND IAC SE
Host2: IAC SB TERMINAL-SPEED IS "1200,1200" IAC SE

(This command is 15 octets.)

Implementation Suggestions

Many systems allow only certain discrete terminal speeds. In such cases it is possible that a speed may be received that does not match one of the allowed values. We suggest that you pick the nearest speed that is allowed, rounding in a "safe" direction. Safety will depend upon the use of the speed information. If it is being used for padding, it is best to round up, since too much padding is better than too little.

RFC-1080 Telnet Remote Flow Control Option

Charles Hedrick; Rutgers University
November 1988

This RFC specifies a standard for the Internet community. Hosts on the Internet that do remote flow control within the Telnet protocol are expected to adopt and implement this standard. Distribution of this memo is unlimited.

Command Name and Code

TOGGLE-FLOW-CONTROL 33

Default

WON'T TOGGLE-FLOW-CONTROL
DON'T TOGGLE-FLOW-CONTROL

Flow control information will not be exchanged.

RFC-1080 Telnet Remote Flow Control Option

Command Meanings

IAC WILL TOGGLE-FLOW-CONTROL

Sender is willing to enable and disable flow control upon command.

IAC WON'T TOGGLE-FLOW-CONTROL

Sender refuses to enable and disable flow control. Nothing is implied about whether sender does or does not use flow control. It is simply unwilling to enable and disable it using this protocol.

IAC DO TOGGLE-FLOW-CONTROL

Sender is willing to send commands to enable and disable flow control.

IAC DON'T TOGGLE-FLOW-CONTROL

Sender refuses to send command to enable and disable flow control.

IAC SB TOGGLE-FLOW-CONTROL OFF IAC SE

Sender requests receiver to disable flow control. The code for OFF is 0.

IAC SB TOGGLE-FLOW-CONTROL ON IAC SE

Sender requests receiver to enable flow control. The code for ON is 1.

RFC-1080 Telnet Remote Flow Control Option

Motivation

This memo describes a method of remotely toggling flow control between a user telnet process and the attached terminal. Only flow control of data being transmitted from the telnet process to the terminal is considered. Many systems will also allow flow control of data from the terminal to the telnet process. However there is seldom need to change this behavior repeatedly during the session.

There are two common ways of doing flow control: hardware and software. Hardware flow control uses signals on wires dedicated for this purpose. Software flow control uses one or two specific characters sent along the same path as normal input data. Most commonly, XOFF (control-S) and XON (control-Q) are used to stop and start output, respectively. The option described herein is useful primarily where software flow control is being used. (Since hardware flow control does not preempt any characters, there is normally no need to disable it.)

The primary difficulty with software flow control is that it preempts one or two characters. Host software often requires the user to be able to input every possible ASCII character. (Certain editors are notorious for having XOFF and XON as commonly-used commands.) For this reason, operating systems often allow programs to disable flow control. While it is disabled, the characters that normally signal flow control may be read as normal input. In a telnet environment, flow control is normally done by the user telnet process, not by the host computer. Thus this RFC defines a way to propagate flow control status from the host computer to the user telnet process.

RFC-1080 Telnet Remote Flow Control Option

Description of the Option

Use of the option requires two phases. In the first phase, the telnet processes agree that one of them will TOGGLE-FLOW-CONTROL. WILL and DO are used only in this first phase. In general there will be only one exchange of WILL and DO for a session. Subnegotiations must not be issued until DO and WILL have been exchanged. It is permissible for either side to turn off the option by sending a WONT or DONT. Should this happen, no more subnegotiations may be sent, unless the option is reenabled by another exchange of DO and WILL.

Once the hosts have exchanged a WILL and a DO, the sender of the DO TOGGLE-FLOW-CONTROL is free to send subnegotiations to enable and disable flow control in the other process. Normally, the sender of the DO will be a host, and the other end will be a user telnet process, which is connected to a terminal. Thus the protocol is normally asymmetric. However it may be used in both directions without confusion should need for this arise.

As soon as the DO and WILL have been exchanged, the sender of the WILL must enable flow control. This allows flow control to begin in a known state. Should the option be disabled by exchange of DONT and WONT, flow control may revert to an implementation-defined default state. It is not safe to assume that flow control will remain in the state requested by the most recent subnegotiation.

Currently, only two command codes are defined for the subnegotiations: flow control off (code 0) and flow control on (code 1). Neither of these codes requires any additional data. However it is possible that additional commands may be added. Thus subnegotiations having command codes other than 0 and 1 should be ignored.

Here is an example of use of this option:

Host1: IAC DO TOGGLE-FLOW-CONTROL
Host2: IAC WILL TOGGLE-FLOW-CONTROL

(Host1 is now free to send commands to change flow control. Note that host2 must now have enabled flow control.)

Host1: IAC SB TOGGLE-FLOW-CONTROL OFF IAC SE
Host1: IAC SB TOGGLE-FLOW-CONTROL ON IAC SE

RFC-1084 BOOTP Vendor Information Extensions

J. Reynolds
USC/Information Sciences Institute
December 1988

This memo describes an addition to the Bootstrap Protocol (BOOTP). Comments and suggestions for improvements are sought. Distribution of this memo is unlimited. This version incorporates all additions through March 1990.

RFC-1084 BOOTP Vendor Information Extensions

Introduction

This RFC is a slight revision and extension of RFC-1048 by Philip Prindeville, who should be credited with the original work in this memo. This memo will be updated as additional tags are are defined. This edition introduces Tag 13 for Boot File Size and Tag 14 for Merit Dump File.

As workstations and personal computers proliferate on the Internet, the administrative complexity of maintaining a network is increased by an order of magnitude. The assignment of local network resources to each client represents one such difficulty. In most environments, delegating such responsibility to the user is not plausible and, indeed, the solution is to define the resources in uniform terms, and to automate their assignment.

The basic Bootstrap Protocol dealt with the issue of assigning an internet address to a client, as well as a few other resources. The protocol included provisions for vendor-defined resource information.

This memo defines a (potentially) vendor-independent interpretation of this resource information.

RFC-1084 BOOTP Vendor Information Extensions

Overview of BOOTP

While the Reverse Address Resolution Protocol (RARP) may be used to assign an IP address to a local network hardware address, it provides only part of the functionality needed. Though this protocol can be used in conjunction with other supplemental protocols (the Resource Location Protocol [RFC-887], the Domain Name System), a more integrated solution may be desirable.

Bootstrap Protocol (BOOTP) is a UDP/IP-based protocol that allows a booting host to configure itself dynamically, and more significantly, without user supervision. It provides a means to assign a host its IP address, a file from which to download a boot program from some server, that server's address, and (if present) the address of an Internet gateway.

One obvious advantage of this procedure is the centralized management of network addresses, which eliminates the need for per-host unique configuration files. In an environment with several hundred hosts, maintaining local configuration information and operating system versions specific to each host might otherwise become chaotic. By categorizing hosts into classes and maintaining configuration information and boot programs for each class, the complexity of this chore may be reduced in magnitude.

RFC-1084 BOOTP Vendor Information Extensions

BOOTP Vendor Information Format

The full description of the BOOTP request/reply packet format may be found in [RFC-951]. The rest of this document will concern itself with the last field of the packet, a 64 octet area reserved for vendor information, to be used in a hitherto unspecified fashion. A generalized use of this area for giving information useful to a wide class of machines, operating systems, and configurations follows. In situations where a single BOOTP server is to be used among heterogeneous clients in a single site, a generic class of data may be used.

Vendor Information "Magic Cookie"

As suggested in [RFC-951], the first four bytes of the vendor field have been assigned to the magic cookie, which identifies the mode in which the succeeding data is to be interpreted. The value of the magic cookie is the 4 octet dotted decimal 99.130.83.99 (or hexadecimal number 63.82.53.63) in network byte order.

Format of Individual Fields

The vendor information field has been implemented as a free format, with extendable tagged sub-fields. These sub-fields are length tagged (with exceptions; see below), allowing clients not implementing certain types to correctly skip fields they cannot interpret. Lengths are exclusive of the tag and length octets; all multi-byte quantities are in network byte-order.

RFC-1084 BOOTP Vendor Information Extensions

Fixed Length Data

The fixed length data are comprised of two formats. Those that have no data consist of a single tag octet and are implicitly of one-octet length, while those that contain data consist of one tag octet, one length octet, and length octets of data.

Tag 0: **Pad Field**
Tag 1: **Subnet Mask Field**
Tag 2: **Time Offset Field**
Tag 255: **End Field**

RFC-1084 BOOTP Vendor Information Extensions - Fixed Length Data

Pad Field (Tag: 0, Data: None)

May be used to align subsequent fields to word boundaries required by the target machine (i.e., 32-bit quantities such as IP addresses on 32-bit boundaries).

RFC-1084 BOOTP Vendor Information Extensions - Fixed Length Data

Subnet Mask Field (Tag: 1, Data: 4 subnet mask bytes)

Specifies the net and local subnet mask as per "Internet Standard Subnetting Procedure" [RFC-950]. For convenience, this field must precede the GATEWAY field (below), if present.

RFC-1084 BOOTP Vendor Information Extensions - Fixed Length Data

Time Offset Field (Tag: 2, Data: 4 time offset bytes)

Specifies the time offset of the local subnet in seconds from Coordinated Universal Time (UTC); signed 32-bit integer.

RFC-1084 BOOTP Vendor Information Extensions - Fixed Length Data

End Field (Tag: 255, Data: None)

Specifies end of usable data in the vendor information area. The rest of this field should be filled with PAD zero) octets.

RFC-1084 BOOTP Vendor Information Extensions

Variable Length Data

The variable length data has a single format; it consists of one tag octet, one length octet, and length octets of data.

Tag 3: **Gateway Field**
Tag 4: **Time Server Field**
Tag 5: **IEN-116 Name Server Field**
Tag 6: **Domain Name Server Field**
Tag 7: **Log Server Field**
Tag 8: **Cookie/Quote Server Field**
Tag 9: **LPR Server Field**
Tag 10: **Impress Server Field**
Tag 11: **RLP Server Field**
Tag 12: **Hostname**
Tag 13: **Boot File Size**
Tag 14: **Merit Dump File**
Tag 15-127: Unassigned
Tag 128-254: **Reserved Fields**

RFC-1084 BOOTP Vendor Information Extensions - Variable Length Data

Gateway Field (Tag: 3, Data: N address bytes)

Specifies the IP addresses of N/4 gateways for this subnet. If one of many gateways is preferred, that should be first.

RFC-1084 BOOTP Vendor Information Extensions - Variable Length Data

Time Server Field (Tag: 4, Data: N address bytes)

Specifies the IP addresses of N/4 time servers [RFC-868].

RFC-1084 BOOTP Vendor Information Extensions - Variable Length Data

IEN-116 Name Server Field (Tag: 5, Data: N address bytes)

Specifies the IP addresses of N/4 name servers [IEN-116].

RFC-1084 BOOTP Vendor Information Extensions - Variable Length Data

Domain Name Server Field (Tag: 6, Data: N address bytes)

Specifies the IP addresses of N/4 domain name servers.

RFC-1084 BOOTP Vendor Information Extensions - Variable Length Data

Log Server Field (Tag: 7, Data: N address bytes)

Specifies the IP addresses of N/4 MIT-LCS UDP log server [LOGGING].

RFC-1084 BOOTP Vendor Information Extensions - Variable Length Data

Cookie/Quote Server Field (Tag: 8, Data: N address bytes)

Specifies the IP addresses of N/4 Quote of the Day servers [RFC-865].

RFC-1084 BOOTP Vendor Information Extensions - Variable Length Data

LPR Server Field (Tag: 9, Data: N address bytes)

Specifies the IP addresses of N/4 Berkeley 4BSD printer servers [LPD].

RFC-1084 BOOTP Vendor Information Extensions - Variable Length Data

Impress Server Field (Tag: 10, Data: N address bytes)

Specifies the IP addresses of N/4 Impress network image servers [IMAGEN].

RFC-1084 BOOTP Vendor Information Extensions - Variable Length Data

RLP Server Field (Tag: 11, Data: N address bytes)

Specifies the IP addresses of N/4 Resource Location Protocol (RLP) servers [RFC-887].

RFC-1084 BOOTP Vendor Information Extensions - Variable Length Data

Hostname (Tag: 12, Data: N bytes of hostname)

Specifies the name of the client. The name may or may not domain qualified: this is a site-specific issue.

Boot File Size (Tag: 13, Data: 2)

A two octet value (in network order) specifying the number of 512 octet blocks in the default boot file. Informs BOOTP client how large the BOOTP file image is.

RFC-1084 BOOTP Vendor Information Extensions - Variable Length Data

Merit Dump Field (Tag 14: Data: ?)

Client to dump and name of the file to dump it to.

RFC-1084 BOOTP Vendor Information Extensions - Variable Length Data

Reserved Fields (Tag: 128-254, Data: N bytes of undefined content)

Specifies additional site-specific information, to be interpreted on an implementation-specific basis. This should follow all data with the preceding generic tags 0-127).

Extensions

Additional generic data fields may be registered by contacting:

Joyce K. Reynolds
USC - Information Sciences Institute
4676 Admiralty Way
Marina del Rey, California 90292-6695
or
by E-mail as: JKREYNOLDS@ISI.EDU
(nic handle JKR1)

Implementation specific use of undefined generic types (those in the range 12-127) may conflict with other implementations, and registration is required.

When selecting information to put into the vendor specific area, care should be taken to not exceed the 64 byte length restriction. Nonessential information (such as host name and quote of the day server) may be excluded, which may later be located with a more appropriate service protocol, such as RLP or the WKS resource-type of the domain name system. Indeed, even RLP servers may be discovered using a broadcast request to locate a local RLP server.

Comparison to Alternative Approaches

Extending BOOTP to provide more configuration information than the minimum required by boot PROMs may not be necessary. Rather than having each module in a host (e.g., the time module, the print spooler, the domain name resolver) broadcast to the BOOTP server to obtain the addresses of required servers, it would be better for each of them to multicast directly to the particular server group of interest, possibly using "expanding ring" multicasts.

The multicast approach has the following advantages over the BOOTP approach:

- It eliminates dependency on a third party (the BOOTP server) that may be temporarily unavailable or whose database may be incorrect or incomplete. Multicasting directly to the desired services will locate those servers that are currently available, and only those.
- It reduces the administrative chore of keeping the (probably replicated) BOOTP database up-to-date and consistent. This is especially important in an environment with a growing number of services and an evolving population of servers.
- In some cases, it reduces the amount of packet traffic and/or the delay required to get the desired information. For example, the current time can be obtained by a single multicast to a time server group which evokes replies from those time servers that are currently up. The BOOTP approach would require a broadcast to the BOOTP server, a reply from the BOOTP server, one or more unicasts to time servers (perhaps waiting for long timeouts if the initially chosen server(s) are down), and finally a reply from a server.

One apparent advantage of the proposed BOOTP extensions is that they provide a uniform way to locate servers. However, the multicast approach could also be implemented in a consistent way across multiple services. The V System naming protocol is a good example of this; character string pathnames are used to name any number of resources (i.e., not just files) and a standard subroutine library looks after multicasting to locate the resources, caching the discovered locations, and detecting stale cache data.

Another apparent advantage of the BOOTP approach is that it allows an administrator to easily control which hosts use which servers. The multicast approach favors more distributed control over resource allocation, where each server decides which hosts it will serve, using whatever level of authentication is appropriate for the particular service. For example, time servers usually don't care who they serve (i.e., administrative control via the BOOTP database is unnecessary), whereas file servers usually require strong authentication (i.e., administrative control via the BOOTP database is insufficient).

The main drawback of the multicast approach, of course, is that IP multicasting is not widely implemented, and there is a need to locate existing services which do not understand IP multicasts.

The BOOTP approach may be most efficient in the case that all the information needed by the client host is returned by a single BOOTP reply and each program module simply reads the information it needs from a local table filled in by the BOOTP reply.

Acknowledgments

The following people provided helpful comments on the first edition of this memo: Drew Perkins, of Carnagie Mellon University, Bill Croft, of Stanford University, and co-author of BOOTP, and Steve Deering, also of Stanford University, for contributing the "Comparison to Alternative Approaches" section.

References

[RFC-951] Croft, B., and J. Gilmore, "Bootstrap Protocol", Network Information Center, SRI International, Menlo Park, California, September 1985.

[RFC-903] Finlayson, R., T. Mann, J. Mogul, and M. Theimer, "A Reverse Address Resolution Protocol", Network Information Center, SRI International, Menlo Park, California, June 1984.

[RFC-887] Accetta, M., "Resource Location Protocol", Network Information Center, SRI International, Menlo Park, California, December 1983.

[RFC-1034] Mockapetris, P., "Domain Names - Concepts and Facilities", Network Information Center, SRI International, Menlo Park, California, November 1987.

[RFC-950] Mogul, J., "Internet Standard Subnetting Procedure", Network Information Center, SRI International, Menlo Park, California, August 1985.

[RFC-868] Postel, J., "Time Protocol", Network Information Center, SRI International, Menlo Park, California, May 1983.

[IEN-116] Postel, J., "Internet Name Server", Network Information Center, SRI International, Menlo Park, California, August 1979.

[LOGGING] Clark, D., Logging and Status Protocol", Massachusetts Institute of Technology Laboratory for Computer Science Cambridge, Massachusetts, 1981.

[RFC-865] Postel, J., "Quote of the Day Protocol", Network Information Center, SRI International, Menlo Park, California, May 1983.

[LPD] Campbell, R., "4.2BSD Line Printer Spooler Manual", UNIX Programmer's Manual, Vol II, University of California at Berkeley, Computer Science Division, July 1983.

[IMAGEN] "Image Server XT Programmer's Guide", Imagen Corporation, Santa Clara, California, August 1986.

RFC-1088 A Standard for the Transmission of IP Datagrams over NetBIOS Networks

Leo J. McLaughlin III
The Wollongong Group
February 1989

Status of this Memo

This document specifies a standard method of encapsulating the Internet Protocol [RFC-791] (IP) datagrams on NetBIOS [2] networks. Distribution of this memo is unlimited.

Acknowledgements

This document would not have been possible without the efforts of John Bartas, James Davidson, and Dan Ladermann in the early design and implementation of IP over NetBIOS.

RFC-1088 A Standard for the Transmission of IP Datagrams over NetBIOS Networks

Introduction

The goal of this specification is to allow compatible and interoperable implementations for transmitting IP datagrams over NetBIOS networks.

NetBIOS is a standard which specifies a means of creating virtual circuits and of transmitting and receiving point-to-point, multicast, and broadcast datagrams. This specification uses only the datagram services.

Previous versions of this memo specified the use of the NetBIOS broadcast datagram services instead of the NetBIOS group name services to implement IP broadcasting. These versions are now obsolete.

RFC-1088 A Standard for the Transmission of IP Datagrams over NetBIOS Networks

Description

NetBIOS networks may be used to support IP networks and subnets [RFC-950] of any class. By means of encapsulating IP datagrams within NetBIOS datagrams and assigning IP numbers to the hosts on a NetBIOS network, IP-based applications are supported on these hosts. The addition of a router capable of encapsulating IP packets within ordinary data-link protocols (such as 802.3 [RFC-1042]) as well as within NetBIOS datagrams allows these NetBIOS hosts to communicate with the Internet at large.

RFC-1088 A Standard for the Transmission of IP Datagrams over NetBIOS Networks

Address Mappings

In general, NetBIOS names may be any series of 16 bytes, however, a few values are reserved or used by common networking packages. NetBIOS names for the IP applications on each host are chosen on the basis of the internet number of that host. Since NetBIOS names are a mapping of IP addresses, no physical address query mechanism (e.g., ARP [RFC-826]) is required.

For these internet protocol applications, IP.XX.XX.XX.XX is the NetBIOS name for any IP over NetBIOS host where XX represents the ascii hexadecimal representation of that byte of the internet address.

This addressing scheme allows for the multiplexing of standard datagram protocols over NetBIOS as well as easy visual confirmation of the correctness of a given packet's address.

RFC-1088 A Standard for the Transmission of IP Datagrams over NetBIOS Networks

Broadcast and Multicast Addresses

Broadcast Internet addresses are represented by the NetBIOS group name IP.FF.FF.FF.FF. Currently, no attempt is made to provide support of IP multicast addresses using NetBIOS group names.

RFC-1088 A Standard for the Transmission of IP Datagrams over NetBIOS Networks

Maximum Transmission Unit

The maximum data size of a NetBIOS datagram, and therefore the Maximum Transmission Unit (MTU) for IP over NetBIOS networks, is 512 bytes. Therefore, any hosts communicating with a host on a NetBIOS network may be required to reassemble fragmented datagrams.

RFC-1088 A Standard for the Transmission of IP Datagrams over NetBIOS Networks

Implementation

To support IP on a NetBIOS host for any given IP address the initialization code must:

1) Add IP.XX.XX.XX.XX to the host's NetBIOS name table.
2) Add IP.FF.FF.FF.FF to the host's NetBIOS group name table.
3) Submit a receive datagram request for the reception of NetBIOS datagrams destined for IP.XX.XX.XX.XX.
4) Submit a receive datagram request for the reception of NetBIOS datagrams destined for IP.FF.FF.FF.FF.

When a NetBIOS datagram to either address is received, it is processed by the protocol stack and another receive datagram request is submitted.

When an IP datagram is sent, it is considered to be NetBIOS datagram data and sent by a send datagram request to either IP.XX.XX.XX.XX or IP.FF.FF.FF.FF.

Optionally, the IP software may desire to make adapter status queries of the NetBIOS network. As support for SNMP becomes a requirement for IP hosts, these adapter status queries may become mandatory.

Finally, when the IP support for a given NetBIOS host is discontinued, a cancel command request should be submitted for every pending receive datagram, and a delete name request should be submitted for both the IP.XX.XX.XX.XX and IP.FF.FF.FF.FF address added during initialization.

RFC-1091 Telnet Terminal-Type Option

J. VanBokkelen
FTP Software, Inc.
February 1989

Status of This Memo

This RFC specifies a standard for the Internet community. Hosts on the Internet that exchange terminal type information within the Telnet protocol are expected to adopt and implement this standard.

This standard supersedes RFC-930. A change is made to permit cycling through a list of possible terminal types and selecting the most appropriate. Distribution of this memo is unlimited.

Command Name and Code

TERMINAL-TYPE 24

Default

WON'T TERMINAL-TYPE
DON'T TERMINAL-TYPE

Terminal type information will not be exchanged.

Command Meanings

IAC WILL TERMINAL-TYPE

Sender is willing to send terminal type information in a subsequent sub-negotiation.

IAC WON'T TERMINAL-TYPE

Sender refuses to send terminal type information.

IAC DO TERMINAL-TYPE

Sender is willing to receive terminal type information in a subsequent sub-negotiation.

IAC DON'T TERMINAL-TYPE

Sender refuses to accept terminal type information.

IAC SB TERMINAL-TYPE SEND IAC SE

Server requests client to transmit his (the client's) next terminal type, and switch emulation modes (if more than one terminal type is supported). The code for SEND is 1. (See below.)

IAC SB TERMINAL-TYPE IS ... IAC SE

Client is stating the name of his current (or only) terminal type. The code for IS is 0. (See below.)

Motivation for the Option

On most machines with bit-mapped displays (e.g., PCs and graphics workstations) a client terminal emulation program is used to simulate a conventional ASCII terminal. Most of these programs have multiple emulation modes, frequently with widely varying characteristics. Likewise, modern host system software and applications can deal with a variety of terminal types. What is needed is a means for the client to present a list of available terminal emulation modes to the server, from which the server can select the one it prefers (for arbitrary reasons). There is also need for a mechanism to change emulation modes during the course of a session, perhaps according to the needs of applications programs.

Existing terminal-type passing mechanisms within Telnet were not designed with multiple emulation modes in mind. While multiple names are allowed, they are assumed to be synonyms. Emulation mode changes are not defined, and the list of modes can only be scanned once.

This document defines a simple extension to the existing mechanisms, which meets both of the above criteria. It makes one assumption about the behaviour of implementations coded to the previous standard in order to obtain full backwards-compatibility.

RFC-1091 Telnet Terminal-Type Option

Description of the Option

Willingness to exchange terminal-type information is agreed upon via conventional Telnet option negotiation. WILL and DO are used only to obtain and grant permission for future discussion. The actual exchange of status information occurs within option subcommands (IAC SB TERMINAL-TYPE...).

Once the two hosts have exchanged a WILL and a DO, the sender of the DO TERMINAL-TYPE (the server) is free to request type information. Only the server may send requests (IAC SB TERMINAL-TYPE SEND IAC SE) and only the client may transmit actual type information (within an IAC SB TERMINAL-TYPE IS ... IAC SE command). Terminal type information may not be sent spontaneously, but only in response to a request.

The terminal type information is an NVT ASCII string. Within this string, upper and lower case are considered equivalent. The complete list of valid terminal type names can be found in the latest "Assigned Numbers" RFC [RFC-1060].

The transmission of terminal type information by the Telnet client in response to a query from the Telnet server implies that the client must simultaneously change emulation mode, unless the terminal type sent is a synonym of the preceding terminal type, or there are other prerequisites for entering the new regime (e.g., having agreed upon the Telnet binary option). The receipt of such information by the Telnet server does not imply any immediate change of processing. However, the information may be passed to a process, which may alter the data it sends to suit the particular characteristics of the terminal. For example, some operating systems have a terminal driver that accepts a code indicating the type of terminal being driven. Using the TERMINAL TYPE and BINARY options, a telnet server program on such a system could arrange to have terminals driven as if they were directly connected, including special functions not available to a standard Network Virtual Terminal.

Note that this specification is deliberately asymmetric. It is assumed that server operating systems and applications in general cannot change terminal types at arbitrary points in a session. Thus, the client may only send a new type (and potentially change emulation modes) when the server requests that it do so.

Implementation Issues

The "terminal type" information may be any NVT ASCII string meaningful to both ends of the negotiation. The list of terminal type names in "Assigned Numbers" is intended to minimize confusion caused by alternative "spellings" of the terminal type. For example, confusion would arise if one party were to call a terminal "IBM3278-2" while the other called it "IBM-3278/2". There is no negative acknowledgement for a terminal type that is not understood, but certain other options (such as switching to BINARY mode) may be refused if a valid terminal type name has not been specified.

In some cases, either a particular terminal may be known by more than one name, for example a specific type and a more generic type, or the client may be a workstation with integrated display capable of emulating more than one kind of terminal. In such cases, the sender of the TERMINAL-TYPE IS command should reply to successive TERMINAL- TYPE SEND commands with the various names. In this way, a telnet server that does not understand the first response can prompt for alternatives. If different terminal emulations are supported by the client, the mode of the emulator must be changed to match the last type sent, unless the particular emulation has other Telnet options (e.g., BINARY) as prerequisites (in which case, the emulation will switch to the last type sent when the prerequisite is fulfilled). When types are synonyms, they should be sent in order from most to least specific.

When the server (the receiver of the TERMINAL-TYPE IS) receives the same response two consecutive times, this indicates the end of the list of available types. Similarly, the client should indicate it has sent all available names by repeating the last one sent. If an additional request is received, this indicates that the server (the sender of the IS) wishes to return to the top of the list of available types (probably to select the least of N evils).

Server implementations conforming to the previous standard will cease sending TERMINAL-TYPE SEND commands after receiving the same response two consecutive times, which will work according to the old standard. It is assumed that client implementations conforming to the previous standard will send the last type on the list in response to a third query (as well as the second). New-style servers must recognize this and not send more queries.

The type "UNKNOWN" should be used if the type of the terminal is unknown or unlikely to be recognized by the other party.

The complete and up-to-date list of terminal type names will be maintained in the "Assigned Numbers". The maximum length of a terminal type name is 40 characters.

User Interfaces

Telnet clients and servers conforming to this specification should provide the following functions in their user interfaces:

Clients supporting multiple emulation modes should allow the user to specify which of the modes is preferred (which name is sent first), prior to connection establishment. The order of the names sent cannot be changed after the negotiation has begun. This initial mode will also become the default with servers which do not support TERMINAL TYPE.

Servers should store the current terminal type name and the list of available names in a manner such that they are accessible to both the user (via a keyboard command) and any applications which need the information. In addition, there should be a corresponding mechanism to request a change of terminal types, by initiating a series of SEND/IS sub-negotiations.

Examples

In this example, the server finds the first type acceptable.

Server: IAC DO TERMINAL-TYPE
Client: IAC WILL TERMINAL-TYPE
(Server may now request a terminal type at any time.)
Server: IAC SB TERMINAL-TYPE SEND IAC SE
Client: IAC SB TERMINAL-TYPE IS IBM-3278-2 IAC SE

In this example, the server requests additional terminal types, and accepts the second (and last on the client's list) type sent (RFC 930 compatible):

Server: IAC DO TERMINAL-TYPE
Client: IAC WILL TERMINAL-TYPE
(Server may now request a terminal type at any time.)
Server: IAC SB TERMINAL-TYPE SEND IAC SE
Client: IAC SB TERMINAL-TYPE IS ZENITH-H19 IAC SE
Server: IAC SB TERMINAL-TYPE SEND IAC SE
Client: IAC SB TERMINAL-TYPE IS UNKNOWN IAC SE
Server: IAC SB TERMINAL-TYPE SEND IAC SE
Client: IAC SB TERMINAL-TYPE IS UNKNOWN IAC SE

In this example, the server requests additional terminal types, and proceeds beyond the end-of-list, to select the first type offered by the client (new-type client and server):

Server: IAC DO TERMINAL-TYPE
Client: IAC WILL TERMINAL-TYPE
(Server may now request a terminal type at any time.)
Server: IAC SB TERMINAL-TYPE SEND IAC SE
Client: IAC SB TERMINAL-TYPE IS DEC-VT220 IAC SE
Server: IAC SB TERMINAL-TYPE SEND IAC SE
Client: IAC SB TERMINAL-TYPE IS DEC-VT100 IAC SE
Server: IAC SB TERMINAL-TYPE SEND IAC SE
Client: IAC SB TERMINAL-TYPE IS DEC-VT52 IAC SE
Server: IAC SB TERMINAL-TYPE SEND IAC SE
Client: IAC SB TERMINAL-TYPE IS DEC-VT52 IAC SE
Server: IAC SB TERMINAL-TYPE SEND IAC SE
Client: IAC SB TERMINAL-TYPE IS DEC-VT220 IAC SE

Reviser's note:

I owe much of this text to RFC-884 and RFC-930, by Marvin Solomon and Edward Wimmers of the University of Wisconsin - Madison, and I owe the idea of the extension to discussions on the "tn3270" mailing list in the Summer of 1987.

Author's Address

James VanBokkelen
FTP Software, Inc.
26 Princess Street
Wakefield, MA 01880-3004

Phone: (617) 246-0900
Email: jbvb@ftp.com

9. References:

[1] Postel, J., and J. Reynolds, "Telnet Protocol Specification", RFC 854, USC Information Sciences Institute, May 1983.

[2] Postel, J., and J. Reynolds, "Telnet Option Specification", RFC 855, USC Information Sciences Institute, May 1983.

[3] Solomon, M., and E. Wimmers, "Telnet Terminal Type Option", RFC 930, University of Wisconsin - Madison, January 1985.

[4] Reynolds, J., and J. Postel, "Assigned Numbers", RFC 1010, USC Information Sciences Institute, May 1987.

RFC-1094 NFS: Network File System Protocol Specification

Network Working Group
Sun Microsystems, Inc.
March 1989

This RFC describes a protocol that Sun Microsystems, Inc., and others are using. A new version of the protocol is under development, but others may benefit from the descriptions of the current protocol, and discussion of some of the design issues. Distribution of this memo is unlimited.

RFC-1094 Network File System

Introduction

The Sun Network File System (NFS) protocol provides transparent remote access to shared files across networks. The NFS protocol is designed to be portable across different machines, operating systems, network architectures, and transport protocols. This portability is achieved through the use of Remote Procedure Call (RPC) primitives built on top of an eXternal Data Representation (XDR). Implementations already exist for a variety of machines, from personal computers to supercomputers.

The supporting mount protocol allows the server to hand out remote access privileges to a restricted set of clients. It performs the operating system-specific functions that allow, for example, to attach remote directory trees to some local file system.

Remote Procedure Call
External Data Representation
Stateless Servers

Remote Procedure Call

Sun's Remote Procedure Call specification provides a procedure- oriented interface to remote services. Each server supplies a "program" that is a set of procedures. NFS is one such program. The combination of host address, program number, and procedure number specifies one remote procedure. A goal of NFS was to not require any specific level of reliability from its lower levels, so it could potentially be used on many underlying transport protocols, or even another remote procedure call implementation. For ease of discussion, the rest of this document will assume NFS is implemented on top of Sun RPC, described in RFC-1057, "RPC: Remote Procedure Call Protocol Specification".

External Data Representation

The eXternal Data Representation (XDR) standard provides a common way of representing a set of data types over a network. The NFS Protocol Specification is written using the RPC data description language. For more information, see RFC-1014, "XDR: External Data Representation Standard". Although automated RPC/XDR compilers exist to generate server and client "stubs", NFS does not require their use. Any software that provides equivalent functionality can be used, and if the encoding is exactly the same it can interoperate with other implementations of NFS.

Stateless Servers

The NFS protocol was intended to be as stateless as possible. That is, a server should not need to maintain any protocol state information about any of its clients in order to function correctly. Stateless servers have a distinct advantage over stateful servers in the event of a failure. With stateless servers, a client need only retry a request until the server responds; it does not even need to know that the server has crashed, or the network temporarily went down. The client of a stateful server, on the other hand, needs to either detect a server failure and rebuild the server's state when it comes back up, or cause client operations to fail.

This may not sound like an important issue, but it affects the protocol in some unexpected ways. We feel that it may be worth a bit of extra complexity in the protocol to be able to write very simple servers that do not require fancy crash recovery. Note that even if a so-called "reliable" transport protocol such as TCP is used, the client must still be able to handle interruptions of service by re- opening connections when they time out. Thus, a stateless protocol may actually simplify the implementation.

On the other hand, NFS deals with objects such as files and directories that inherently have state -- what good would a file be if it did not keep its contents intact? The goal was to not introduce any extra state in the protocol itself. Inherently stateful operations such as file or record locking, and remote execution, were implemented as separate services, not described in this document.

The basic way to simplify recovery was to make operations as "idempotent" as possible (so that they can potentially be repeated). Some operations in this version of the protocol did not attain this goal; luckily most of the operations (such as Read and Write) are idempotent. Also, most server failures occur between operations, not between the receipt of an operation and the response. Finally, although actual server failures may be rare, in complex networks, failures of any network, router, or bridge may be indistinguishable from a server failure.

RFC-1094 Network File System

NFS PROTOCOL DEFINITION

Servers change over time, and so can the protocol that they use. RPC provides a version number with each RPC request. This RFC describes version two of the NFS protocol. Even in the second version, there are a few obsolete procedures and parameters, which will be removed in later versions. An RFC for version three of the NFS protocol is currently under preparation.

RFC-1094 Network File System: Protocol Definition

File System Model

NFS assumes a file system that is hierarchical, with directories as all but the bottom level of files. Each entry in a directory (file, directory, device, etc.) has a string name. Different operating systems may have restrictions on the depth of the tree or the names used, as well as using different syntax to represent the "pathname", which is the concatenation of all the "components" (directory and file names) in the name. A "file system" is a tree on a single server (usually a single disk or physical partition) with a specified "root". Some operating systems provide a "mount" operation to make all file systems appear as a single tree, while others maintain a "forest" of file systems. Files are unstructured streams of uninterpreted bytes. Version 3 of NFS uses slightly more general file system model.

NFS looks up one component of a pathname at a time. It may not be obvious why it does not just take the whole pathname, traipse down the directories, and return a file handle when it is done. There are several good reasons not to do this. First, pathnames need separators between the directory components, and different operating systems use different separators. We could define a Network Standard Pathname Representation, but then every pathname would have to be parsed and converted at each end. Other issues are discussed in NFS Implementation Issues.

Although files and directories are similar objects in many ways, different procedures are used to read directories and files. This provides a network standard format for representing directories. The same argument as above could have been used to justify a procedure that returns only one directory entry per call. The problem is efficiency. Directories can contain many entries, and a remote call to return each would be just too slow.

Server Procedures

The protocol definition is given as a set of procedures with arguments and results defined using the RPC language (XDR language extended with program, version, and procedure declarations). A brief description of the function of each procedure should provide enough information to allow implementation. The section on basic data types provides more detail.

All of the procedures in the NFS protocol are assumed to be synchronous. When a procedure returns to the client, the client can assume that the operation has completed and any data associated with the request is now on stable storage. For example, a client WRITE request may cause the server to update data blocks, file system information blocks (such as indirect blocks), and file attribute information (size and modify times). When the WRITE returns to the client, it can assume that the write is safe, even in case of a server crash, and it can discard the data written. This is a very important part of the statelessness of the server. If the server waited to flush data from remote requests, the client would have to save those requests so that it could resend them in case of a server crash.

```
/*
* Remote file service routines
*/
program NFS_PROGRAM {
        version NFS_VERSION {
                void
                NFSPROC_NULL(void)              = 0;

                attrstat
                NFSPROC_GETATTR(fhandle)        = 1;

                attrstat
                NFSPROC_SETATTR(sattrargs)      = 2;

                void
                NFSPROC_ROOT(void)              = 3;

                diropres
                NFSPROC_LOOKUP(diropargs)       = 4;

                readlinkres
                NFSPROC_READLINK(fhandle)       = 5;

                readres
                NFSPROC_READ(readargs)          = 6;

                void
                NFSPROC_WRITECACHE(void)        = 7;

                attrstat
                NFSPROC_WRITE(writeargs)        = 8;

                diropres
                NFSPROC_CREATE(createargs)      = 9;

                stat
```

```
                NFSPROC_REMOVE(diropargs)       = 10;

                stat
                NFSPROC_RENAME(renameargs)      = 11;

                stat
                NFSPROC_LINK(linkargs)          = 12;

                stat
                NFSPROC_SYMLINK(symlinkargs)    = 13;

                diropres
                NFSPROC_MKDIR(createargs)       = 14;

                stat
                NFSPROC_RMDIR(diropargs)         = 15;

                readdirres
                NFSPROC_READDIR(readdirargs) = 16;

                statfsres
                NFSPROC_STATFS(fhandle)         = 17;
        } = 2;
} = 100003;
```

Do Nothing

```
void
NFSPROC_NULL(void) = 0;
```

This procedure does no work. It is made available in all RPC services to allow server response testing and timing.

Get File Attributes

```
attrstat
NFSPROC_GETATTR (fhandle) = 1;
```

If the reply status is `NFS_OK`, then the reply attributes contains the attributes for the file given by the input fhandle.

Set File Attributes

```
struct sattrargs {
        fhandle file;
        sattr attributes;
};

attrstat
NFSPROC_SETATTR (sattrargs) = 2;
```

The "attributes" argument contains fields which are either -1 or are the new value for the attributes of "file". If the reply status is `NFS_OK`, then the reply attributes have the attributes of the file after the "`SETATTR`" operation has completed.

Notes: The use of -1 to indicate an unused field in "attributes" is changed in the next version of the protocol.

Get Filesystem Root

```
void
NFSPROC_ROOT(void) = 3;
```

Obsolete. This procedure is no longer used because finding the root file handle of a filesystem requires moving pathnames between client and server. To do this right, we would have to define a network standard representation of pathnames. Instead, the function of looking up the root file handle is done by the `MNTPROC_MNT` procedure. (See the Appendix "Mount Protocol Definition", for details).

Look Up File Name

```
        diropres
        NFSPROC_LOOKUP(diropargs) = 4;
```

If the reply "status" is `NFS_OK`, then the reply "file" and reply "attributes" are the file handle and attributes for the file "name" in the directory given by "dir" in the argument.

Read From Symbolic Link

```
union readlinkres switch (stat status) {
case NFS_OK:
    path data;
default:
    void;
};

readlinkres
NFSPROC_READLINK(fhandle) = 5;
```

If "status" has the value `NFS_OK`, then the reply "data" is the data in the symbolic link given by the file referred to by the fhandle argument.

Notes: Since NFS always parses pathnames on the client, the pathname in a symbolic link may mean something different (or be meaningless) on a different client or on the server if a different pathname syntax is used.

Read From File

```
struct readargs {
        fhandle file;
        unsigned offset;
        unsigned count;
        unsigned totalcount;
};

union readres switch (stat status) {
case NFS_OK:
        fattr attributes;
        nfsdata data;
default:
        void;
};

readres
NFSPROC_READ(readargs) = 6;
```

Returns up to "count" bytes of "data" from the file given by "file", starting at "offset" bytes from the beginning of the file. The first byte of the file is at offset zero. The file attributes after the read takes place are returned in "attributes".

Notes: The argument "totalcount" is unused, and is removed in the next protocol revision.

Write to Cache

```
void
NFSPROC_WRITECACHE(void) = 7;
```

To be used in the next protocol revision.

Write to File

```
struct writeargs {
        fhandle file;
        unsigned beginoffset;
        unsigned offset;
        unsigned totalcount;
        nfsdata data;
};

attrstat
NFSPROC_WRITE(writeargs) = 8;
```

Writes "data" beginning "offset" bytes from the beginning of "file". The first byte of the file is at offset zero. If the reply "status" is NFS_OK, then the reply "attributes" contains the attributes of the file after the write has completed. The write operation is atomic. Data from this "WRITE" will not be mixed with data from another client's "WRITE".

Notes: The arguments "beginoffset" and "totalcount" are ignored and are removed in the next protocol revision.

Create File

```
struct createargs {
        diropargs where;
        sattr attributes;
};

diropres
NFSPROC_CREATE(createargs) = 9;
```

The file "name" is created in the directory given by "dir". The initial attributes of the new file are given by "attributes". A reply "status" of `NFS_OK` indicates that the file was created, and reply "file" and reply "attributes" are its file handle and attributes. Any other reply "status" means that the operation failed and no file was created.

Notes: This routine should pass an exclusive create flag, meaning "create the file only if it is not already there".

Remove File

```
stat
NFSPROC_REMOVE(diropargs) = 10;
```

The file "name" is removed from the directory given by "dir". A reply of `NFS_OK` means the directory entry was removed.

Notes: possibly non-idempotent operation.

Rename File

```
struct renameargs {
        diropargs from;
        diropargs to;
};

stat
NFSPROC_RENAME(renameargs) = 11;
```

The existing file "from.name" in the directory given by "from.dir" is renamed to "to.name" in the directory given by "to.dir". If the reply is `NFS_OK`, the file was renamed. The `RENAME` operation is atomic on the server; it cannot be interrupted in the middle.

Notes: possibly non-idempotent operation.

Create Link to File

Procedure 12, Version 2.

```
struct linkargs {
        fhandle from;
        diropargs to;
};

stat
NFSPROC_LINK(linkargs) = 12;
```

Creates the file "to.name" in the directory given by "to.dir", which is a hard link to the existing file given by "from". If the return value is `NFS_OK`, a link was created. Any other return value indicates an error, and the link was not created.

A hard link should have the property that changes to either of the linked files are reflected in both files. When a hard link is made to a file, the attributes for the file should have a value for "nlink" that is one greater than the value before the link.

Notes: possibly non-idempotent operation.

Create Symbolic Link

```
struct symlinkargs {
        diropargs from;
        path to;
        sattr attributes;
};

stat
NFSPROC_SYMLINK(symlinkargs) = 13;
```

Creates the file "from.name" with ftype `NFLNK` in the directory given by "from.dir". The new file contains the pathname "to" and has initial attributes given by "attributes". If the return value is `NFS_OK`, a link was created. Any other return value indicates an error, and the link was not created.

A symbolic link is a pointer to another file. The name given in "to" is not interpreted by the server, only stored in the newly created file. When the client references a file that is a symbolic link, the contents of the symbolic link are normally transparently reinterpreted as a pathname to substitute. A `READLINK` operation returns the data to the client for interpretation.

Notes: On UNIX servers the attributes are never used, since symbolic links always have mode 0777.

Create Directory

```
diropres
NFSPROC_MKDIR (createargs) = 14;
```

The new directory "where.name" is created in the directory given by "where.dir". The initial attributes of the new directory are given by "attributes". A reply "status" of `NFS_OK` indicates that the new directory was created, and reply "file" and reply "attributes" are its file handle and attributes. Any other reply "status" means that the operation failed and no directory was created.

Notes: possibly non-idempotent operation.

Remove Directory

```
stat
NFSPROC_RMDIR(diropargs) = 15;
```

The existing empty directory "name" in the directory given by "dir" is removed. If the reply is `NFS_OK`, the directory was removed.

Notes: possibly non-idempotent operation.

Read From Directory

```
struct readdirargs {
        fhandle dir;
        nfscookie cookie;
        unsigned count;
};

struct entry {
        unsigned fileid;
        filename name;
        nfscookie cookie;
        entry *nextentry;
};

union readdirres switch (stat status) {
case NFS_OK:
        struct {
                entry *entries;
                bool eof;
        } readdirok;
default:
        void;
};

readdirres
NFSPROC_READDIR (readdirargs) = 16;
```

Returns a variable number of directory entries, with a total size of up to "count" bytes, from the directory given by "dir". If the returned value of "status" is `NFS_OK`, then it is followed by a variable number of "entry"s. Each "entry" contains a "fileid" which consists of a unique number to identify the file within a filesystem, the "name" of the file, and a "cookie" which is an opaque pointer to the next entry in the directory. The cookie is used in the next `READDIR` call to get more entries starting at a given point in the directory. The special cookie zero (all bits zero) can be used to get the entries starting at the beginning of the directory. The "fileid" field should be the same number as the "fileid" in the the attributes of the file. (See section "2.3.5. fattr" under "Basic Data Types".) The "eof" flag has a value of TRUE if there are no more entries in the directory.

Get Filesystem Attributes

```
union statfsres (stat status) {
case NFS_OK:
    struct {
        unsigned tsize;
        unsigned bsize;
        unsigned blocks;
        unsigned bfree;
        unsigned bavail;
    } info;
default:
        void;
};

statfsres
NFSPROC_STATFS(fhandle) = 17;
```

If the reply "status" is `NFS_OK`, then the reply "info" gives the attributes for the filesystem that contains file referred to by the input fhandle. The attribute fields contain the following values:

tsize The optimum transfer size of the server in bytes. This is the number of bytes the server would like to have in the data part of READ and WRITE requests.

bsize The block size in bytes of the filesystem.

blocks The total number of "bsize" blocks on the filesystem.

bfree The number of free "bsize" blocks on the filesystem.

bavail The number of "bsize" blocks available to non-privileged users.

Notes: This call does not work well if a filesystem has variable size blocks.

Basic Data Types

The following XDR definitions are basic structures and types used in other structures described further on.

stat
ftype
fhandle
timeval
fatr
satr
filename
path
attrstat
diropargs
diropres1094_Type_diropres

stat

```
enum stat {
    NFS_OK = 0,
    NFSERR_PERM=1,
    NFSERR_NOENT=2,
    NFSERR_IO=5,
    NFSERR_NXIO=6,
    NFSERR_ACCES=13,
    NFSERR_EXIST=17,
    NFSERR_NODEV=19,
    NFSERR_NOTDIR=20,
    NFSERR_ISDIR=21,
    NFSERR_FBIG=27,
    NFSERR_NOSPC=28,
    NFSERR_ROFS=30,
    NFSERR_NAMETOOLONG=63,
    NFSERR_NOTEMPTY=66,
    NFSERR_DQUOT=69,
    NFSERR_STALE=70,
    NFSERR_WFLUSH=99
};
```

The "stat" type is returned with every procedure's results. A value of `NFS_OK` indicates that the call completed successfully and the results are valid. The other values indicate some kind of error occurred on the server side during the servicing of the procedure. The error values are derived from UNIX error numbers.

`NFSERR_PERM`
Not owner. The caller does not have correct ownership to perform the requested operation.

`NFSERR_NOENT`
No such file or directory. The file or directory specified does not exist.

`NFSERR_IO`
Some sort of hard error occurred when the operation was in progress. This could be a disk error, for example.

`NFSERR_NXIO`
No such device or address.

`NFSERR_ACCES`
Permission denied. The caller does not have the correct permission to perform the requested operation.

`NFSERR_EXIST`
File exists. The file specified already exists.

`NFSERR_NODEV`
No such device.

`NFSERR_NOTDIR`

Not a directory. The caller specified a non-directory in a directory operation.

NFSERR_ISDIR
Is a directory. The caller specified a directory in a non- directory operation.

NFSERR_FBIG
File too large. The operation caused a file to grow beyond the server's limit.

NFSERR_NOSPC
No space left on device. The operation caused the server's filesystem to reach its limit.

NFSERR_ROFS
Read-only filesystem. Write attempted on a read-only filesystem.

NFSERR_NAMETOOLONG
File name too long. The file name in an operation was too long.

NFSERR_NOTEMPTY
Directory not empty. Attempted to remove a directory that was not empty.

NFSERR_DQUOT
Disk quota exceeded. The client's disk quota on the server has been exceeded.

NFSERR_STALE
The "fhandle" given in the arguments was invalid. That is, the file referred to by that file handle no longer exists, or access to it has been revoked.

NFSERR_WFLUSH
The server's write cache used in the "WRITECACHE" call got flushed to disk.

ftype

```
enum ftype {
    NFNON = 0,
    NFREG = 1,
    NFDIR = 2,
    NFBLK = 3,
    NFCHR = 4,
    NFLNK = 5
};
```

The enumeration "ftype" gives the type of a file. The type `NFNON` indicates a non-file, `NFREG` is a regular file, `NFDIR` is a directory, `NFBLK` is a block-special device, `NFCHR` is a character-special device, and `NFLNK` is a symbolic link.

fhandle

```
typedef opaque fhandle[FHSIZE];
```

The "fhandle" is the file handle passed between the server and the client. All file operations are done using file handles to refer to a file or directory. The file handle can contain whatever information the server needs to distinguish an individual file.

timeval

```
struct timeval {
    unsigned int seconds;
    unsigned int useconds;
};
```

The "timeval" structure is the number of seconds and microseconds since midnight January 1, 1970, Greenwich Mean Time. It is used to pass time and date information.

fattr

```
struct fattr {
    ftype        type;
    unsigned int mode;
    unsigned int nlink;
    unsigned int uid;
    unsigned int gid;
    unsigned int size;
    unsigned int blocksize;
    unsigned int rdev;
    unsigned int blocks;
    unsigned int fsid;
    unsigned int fileid;
    timeval      atime;
    timeval      mtime;
    timeval      ctime;
};
```

The "fattr" structure contains the attributes of a file; "type" is the type of the file; "nlink" is the number of hard links to the file (the number of different names for the same file); "uid" is the user identification number of the owner of the file; "gid" is the group identification number of the group of the file; "size" is the size in bytes of the file; "blocksize" is the size in bytes of a block of the file; "rdev" is the device number of the file if it is type `NFCHR` or `NFBLK`; "blocks" is the number of blocks the file takes up on disk; "fsid" is the file system identifier for the filesystem containing the file; "fileid" is a number that uniquely identifies the file within its filesystem; "atime" is the time when the file was last accessed for either read or write; "mtime" is the time when the file data was last modified (written); and "ctime" is the time when the status of the file was last changed. Writing to the file also changes "ctime" if the size of the file changes.

> "Mode" is the access mode encoded as a set of bits. Notice that the file type is specified both in the mode bits and in the file type. This is really a bug in the protocol and will be fixed in future versions. The descriptions given below specify the bit positions using octal numbers.
>
> 0040000 This is a directory; "type" field should be `NFDIR`.
> 0020000 This is a character special file; "type" field should be `NFCHR`.
> 0060000 This is a block special file; "type" field should be `NFBLK`.
> 0100000 This is a regular file; "type" field should be `NFREG`.
> 0120000 This is a symbolic link file; "type" field should be `NFLNK`.
> 0140000 This is a named socket; "type" field should be `NFNON`.
> 0004000 Set user id on execution.
> 0002000 Set group id on execution.
> 0001000 Save swapped text even after use.
> 0000400 Read permission for owner.
> 0000200 Write permission for owner.
> 0000100 Execute and search permission for owner.
> 0000040 Read permission for group.
> 0000020 Write permission for group.
> 0000010 Execute and search permission for group.
> 0000004 Read permission for others.

0000002 Write permission for others.
0000001 Execute and search permission for others.

Notes: The bits are the same as the mode bits returned by the stat(2) system call in UNIX. The file type is specified both in the mode bits and in the file type. This is fixed in future versions.

The "rdev" field in the attributes structure is an operating system specific device specifier. It will be removed and generalized in the next revision of the protocol.

sattr

```
struct sattr {
    unsigned int mode;
    unsigned int uid;
    unsigned int gid;
    unsigned int size;
    timeval      atime;
    timeval      mtime;
};
```

The "sattr" structure contains the file attributes which can be set from the client. The fields are the same as for "fattr" above. A "size" of zero means the file should be truncated. A value of -1 indicates a field that should be ignored.

filename

```
typedef string filename<MAXNAMLEN>;
```

The type "filename" is used for passing file names or pathname components.

path

```
typedef string path<MAXPATHLEN>;
```

The type "path" is a pathname. The server considers it as a string with no internal structure, but to the client it is the name of a node in a filesystem tree.

attrstat

```
union attrstat switch (stat status) {
case NFS_OK:
    fattr attributes;
default:
    void;
};
```

The "attrstat" structure is a common procedure result. It contains a "status" and, if the call succeeded, it also contains the attributes of the file on which the operation was done.

diropargs

```
struct diropargs {
    fhandle  dir;
    filename name;
};
```

The "diropargs" structure is used in directory operations. The "fhandle" "dir" is the directory in which to find the file "name". A directory operation is one in which the directory is affected.

diropres

```
union diropres switch (stat status) {
case NFS_OK:
    struct {
        fhandle file;
        fattr   attributes;
    } diropok;
default:
    void;
};
```

The results of a directory operation are returned in a "diropres" structure. If the call succeeded, a new file handle "file" and the "attributes" associated with that file are returned along with the "status".

NFS Implementation Issues

The NFS protocol was designed to allow different operating systems to share files. However, since it was designed in a UNIX environment, many operations have semantics similar to the operations of the UNIX file system. This section discusses some of the implementation- specific details and semantic issues.

Server/Client Relationship

The NFS protocol is designed to allow servers to be as simple and general as possible. Sometimes the simplicity of the server can be a problem, if the client wants to implement complicated filesystem semantics.

For example, some operating systems allow removal of open files. A process can open a file and, while it is open, remove it from the directory. The file can be read and written as long as the process keeps it open, even though the file has no name in the filesystem. It is impossible for a stateless server to implement these semantics. The client can do some tricks such as renaming the file on remove, and only removing it on close. We believe that the server provides enough functionality to implement most file system semantics on the client.

Every NFS client can also potentially be a server, and remote and local mounted filesystems can be freely intermixed. This leads to some interesting problems when a client travels down the directory tree of a remote filesystem and reaches the mount point on the server for another remote filesystem. Allowing the server to follow the second remote mount would require loop detection, server lookup, and user revalidation. Instead, we decided not to let clients cross a server's mount point. When a client does a LOOKUP on a directory on which the server has mounted a filesystem, the client sees the underlying directory instead of the mounted directory.

For example, if a server has a file system called "`/usr`" and mounts another file system on "`/usr/src`", if a client mounts "`/usr`", it does NOT see the mounted version of "`/usr/src`". A client could do remote mounts that match the server's mount points to maintain the server's view. In this example, the client would also have to mount "`/usr/src`" in addition to "`/usr`", even if they are from the same server.

Pathname Interpretation

There are a few complications to the rule that pathnames are always parsed on the client. For example, symbolic links could have different interpretations on different clients. Another common problem for non-UNIX implementations is the special interpretation of the pathname ".." to mean the parent of a given directory. The next revision of the protocol uses an explicit flag to indicate the parent instead.

Permission Issues

The NFS protocol, strictly speaking, does not define the permission checking used by servers. However, it is expected that a server will do normal operating system permission checking using AUTH_UNIX style authentication as the basis of its protection mechanism. The server gets the client's effective "uid", effective "gid", and groups on each call and uses them to check permission. There are various problems with this method that can been resolved in interesting ways.

Using "uid" and "gid" implies that the client and server share the same "uid" list. Every server and client pair must have the same mapping from user to "uid" and from group to "gid". Since every client can also be a server, this tends to imply that the whole network shares the same "uid/gid" space. AUTH_DES (and the next revision of the NFS protocol) uses string names instead of numbers, but there are still complex problems to be solved.

Another problem arises due to the usually stateful open operation. Most operating systems check permission at open time, and then check that the file is open on each read and write request. With stateless servers, the server has no idea that the file is open and must do permission checking on each read and write call. On a local filesystem, a user can open a file and then change the permissions so that no one is allowed to touch it, but will still be able to write to the file because it is open. On a remote filesystem, by contrast, the write would fail. To get around this problem, the server's permission checking algorithm should allow the owner of a file to access it regardless of the permission setting.

A similar problem has to do with paging in from a file over the network. The operating system usually checks for execute permission before opening a file for demand paging, and then reads blocks from the open file. The file may not have read permission, but after it is opened it does not matter. An NFS server can not tell the difference between a normal file read and a demand page-in read. To make this work, the server allows reading of files if the "uid" given in the call has either execute or read permission on the file.

In most operating systems, a particular user (on UNIX, the user ID zero) has access to all files no matter what permission and ownership they have. This "super-user" permission may not be allowed on the server, since anyone who can become super-user on their workstation could gain access to all remote files. The UNIX server by default maps user id 0 to -2 before doing its access checking. This works except for NFS root filesystems, where super-user access cannot be avoided.

RFC-1094 Network File System: Implementation Issues

RPC Information

Authentication
The NFS service uses AUTH_UNIX, AUTH_DES, or AUTH_SHORT style authentication, except in the NULL procedure where AUTH_NONE is also allowed.

Transport Protocols
NFS is supported normally on UDP.

Port Number
The NFS protocol currently uses the UDP port number 2049. This is not an officially assigned port, so later versions of the protocol use the "Portmapping" facility of RPC.

Sizes of XDR Structures

These are the sizes, given in decimal bytes, of various XDR structures used in the protocol:

```
/*
 * The maximum number of bytes of data in a READ or WRITE
 * request.
 */
const MAXDATA = 8192;

/* The maximum number of bytes in a pathname argument. */
const MAXPATHLEN = 1024;

/* The maximum number of bytes in a file name argument. */
const MAXNAMLEN = 255;

/* The size in bytes of the opaque "cookie" passed by READDIR. */
const COOKIESIZE  = 4;

/* The size in bytes of the opaque file handle. */
const FHSIZE = 32;
```

Setting RPC Parameters

Various file system parameters and options should be set at mount time. The mount protocol is described in the appendix below. For example, "Soft" mounts as well as "Hard" mounts are usually both provided. Soft mounted file systems return errors when RPC operations fail (after a given number of optional retransmissions), while hard mounted file systems continue to retransmit forever. The maximum transfer sizes are implementation dependent. For efficient operation over a local network, 8192 bytes of data are normally used. This may result in lower-level fragmentation (such as at the IP level). Since some network interfaces may not allow such packets, for operation over slower-speed networks or hosts, or through gateways, transfer sizes of 512 or 1024 bytes often provide better results.

Clients and servers may need to keep caches of recent operations to help avoid problems with non-idempotent operations. For example, if the transport protocol drops the response for a Remove File operation, upon retransmission the server may return an error code of `NFSERR_NOENT` instead of `NFS_OK`. But if the server keeps around the last operation requested and its result, it could return the proper success code. Of course, the server could be crashed and rebooted between retransmissions, but a small cache (even a single entry) would solve most problems.

Appendix: Mount Protocol Definition

Mount Protocol: Introduction

The mount protocol is separate from, but related to, the NFS protocol. It provides operating system specific services to get the NFS off the ground -- looking up server path names, validating user identity, and checking access permissions. Clients use the mount protocol to get the first file handle, which allows them entry into a remote filesystem.

The mount protocol is kept separate from the NFS protocol to make it easy to plug in new access checking and validation methods without changing the NFS server protocol.

Notice that the protocol definition implies stateful servers because the server maintains a list of client's mount requests. The mount list information is not critical for the correct functioning of either the client or the server. It is intended for advisory use only, for example, to warn possible clients when a server is going down.

Version one of the mount protocol is used with version two of the NFS protocol. The only information communicated between these two protocols is the "fhandle" structure.

RPC Information

Authentication
The mount service uses AUTH_UNIX and AUTH_NONE style authentication only.

Transport Protocols
The mount service is supported on both UDP and TCP.

Port Number
Consult the server's portmapper, described in RFC-1057, "RPC: Remote Procedure Call Protocol Specification", to find the port number on which the mount service is registered.

Sizes of XDR Structures

These are the sizes, given in decimal bytes, of various XDR structures used in the protocol:

```
/* The maximum number of bytes in a pathname argument. */
const MNTPATHLEN = 1024;

/* The maximum number of bytes in a name argument. */
const MNTNAMLEN = 255;

/* The size in bytes of the opaque file handle. */
const FHSIZE = 32;
```

Basic Data Types

This section presents the data types used by the mount protocol. In many cases they are similar to the types used in NFS.

RFC-1094 NFS Appendix--Mount Protocol: Basic Data Types

fhandle

```
typedef opaque fhandle[FHSIZE];
```

The type "fhandle" is the file handle that the server passes to the client. All file operations are done using file handles to refer to a file or directory. The file handle can contain whatever information the server needs to distinguish an individual file.

This is the same as the "fhandle" XDR definition in version 2 of the NFS protocol; see section "fhandle" under "Basic Data Types".

RFC-1094 NFS Appendix--Mount Protocol: Basic Data Types

fhstatus

```
union fhstatus switch (unsigned status) {
case 0:
    fhandle directory;
default:
    void;
}
```

The type "fhstatus" is a union. If a "status" of zero is returned, the call completed successfully, and a file handle for the "directory" follows. A non-zero status indicates some sort of error. In this case, the status is a UNIX error number.

dirpath

```
typedef string dirpath<MNTPATHLEN>;
```

The type "dirpath" is a server pathname of a directory.

RFC-1094 NFS Appendix--Mount Protocol: Basic Data Types

name

```
typedef string name<MNTNAMLEN>;
```

The type "name" is an arbitrary string used for various names.

Server Procedures

The following sections define the RPC procedures supplied by a mount server.

```
/*
 * Protocol description for the mount program
 */
program MOUNTPROG {
        /*
         * Version 1 of the mount protocol used with
         * version 2 of the NFS protocol.
         */
        version MOUNTVERS {

                void
                MOUNTPROC_NULL(void) = 0;

                fhstatus
                MOUNTPROC_MNT(dirpath) = 1;

                mountlist
                MOUNTPROC_DUMP(void) = 2;

                void
                MOUNTPROC_UMNT(dirpath) = 3;

                void
                MOUNTPROC_UMNTALL(void) = 4;

                exportlist
                MOUNTPROC_EXPORT(void)  = 5;
        } = 1;
} = 100005;
```

RFC-1094 NFS Appendix--Mount Protocol: Server Procedures

Do Nothing

```
void
MNTPROC_NULL(void) = 0;
```

This procedure does no work. It is made available in all RPC services to allow server response testing and timing.

Add Mount Entry

```
fhstatus
MNTPROC_MNT(dirpath) = 1;
```

If the reply "status" is 0, then the reply "directory" contains the file handle for the directory "dirname". This file handle may be used in the NFS protocol. This procedure also adds a new entry to the mount list for this client mounting "dirname".

Return Mount Entries

```
struct *mountlist {
        name      hostname;
        dirpath   directory;
        mountlist nextentry;
};

mountlist
MNTPROC_DUMP(void) = 2;
```

Returns the list of remote mounted filesystems. The "mountlist" contains one entry for each "hostname" and "directory" pair.

Remove Mount Entry

```
void
MNTPROC_UMNT(dirpath) = 3;
```

Removes the mount list entry for the input "dirpath".

Remove All Mount Entries

```
void
MNTPROC_UMNTALL(void) = 4;
```

Removes all of the mount list entries for this client.

Return Export List

```
struct *groups {
        name grname;
        groups grnext;
};

struct *exportlist {
        dirpath filesys;
        groups groups;
        exportlist next;
};

exportlist
MNTPROC_EXPORT(void) = 5;
```

Returns a variable number of export list entries. Each entry contains a filesystem name and a list of groups that are allowed to import it. The filesystem name is in "filesys", and the group name is in the list "groups".

Notes: The exportlist should contain more information about the status of the filesystem, such as a read-only flag.

Author's Address:

Bill Nowicki
Sun Microsystems, Inc.
Mail Stop 1-40
2550 Garcia Avenue
Mountain View, CA 94043

Phone: (415) 336-7278

Email: nowicki@SUN.COM

RFC-1101 DNS Encoding of Network Names and Other Types

Updates: RFCs 1034, 1035
P. Mockapetris
ISI
April 1989

This RFC proposes two extensions to the Domain Name System:

- A specific method for entering and retrieving RRs which map between network names and numbers.
- Ideas for a general method for describing mappings between arbitrary identifiers and numbers.

The method for mapping between network names and addresses is a proposed standard, the ideas for a general method are experimental.

This RFC assumes that the reader is familiar with the DNS and its use. The data shown is for pedagogical use and does not necessarily reflect the real Internet.

RFC-1101 DNS Encoding of Network Names and Other Types

Introduction

The DNS is extensible and can be used for a virtually unlimited number of data types, name spaces, etc. New type definitions are occasionally necessary as are revisions or deletions of old types (e.g., MX replacement of MD and MF (RFC-974]), and changes described in RFC-973. This RFC describes changes due to the general need to map between identifiers and values, and a specific need for network name support.

Users wish to be able to use the DNS to map between network names and numbers. This need is the only capability found in HOSTS.TXT which is not available from the DNS. In designing a method to do this, there were two major areas of concern:

- Several tradeoffs involving control of network names, the syntax of network names, backward compatibility, etc.

- A desire to create a method which would be sufficiently general to set a good precedent for future mappings, for example, between TCP-port names and numbers, autonomous system names and numbers, X.500 Relative Distinguished Names (RDNs) and their servers, or whatever.

It was impossible to reconcile these two areas of concern for network names because of the desire to unify network number support within existing IP address to host name support. The existing support is the IN-ADDR.ARPA section of the DNS name space. As a result this RFC describes one structure for network names which builds on the existing support for host names, and another family of structures for future yellow pages (YP) functions such as conversions between TCP- port numbers and mnemonics.

Both structures are described in following sections. Each structure has a discussion of design issues and specific structure recommendations.

We wish to avoid defining structures and methods which can work but do not because of indifference or errors on the part of system administrators when maintaining the database. The WKS RR is an example. Thus, while we favor distribution as a general method, we also recognize that centrally maintained tables (such as HOSTS.TXT) are usually more consistent though less maintainable and timely. Hence we recommend both specific methods for mapping network names, addresses, and subnets, as well as an instance of the general method for mapping between allocated network numbers and network names. (Allocation is centrally performed by the SRI Network Information Center, aka the NIC).

Network Name Issues and Discussion

The issues involved in the design were the definition of network name syntax, the mappings to be provided, and possible support for similar functions at the subnet level.

RFC-1101 DNS Encoding of Network Names and Other Types

Network Name Syntax

The current syntax for network names, as defined by (RFC-952) is an alphanumeric string of up to 24 characters, which begins with an alpha, and may include "." and "-" except as first and last characters. This is the format which was also used for host names before the DNS. Upward compatibility with existing names might be a goal of any new scheme.

However, the present syntax has been used to define a flat name space, and hence would prohibit the same distributed name allocation method used for host names. There is some sentiment for allowing the NIC to continue to allocate and regulate network names, much as it allocates numbers, but the majority opinion favors local control of network names. Although it would be possible to provide a flat space or a name space in which, for example, the last label of a domain name captured the old-style network name, any such approach would add complexity to the method and create different rules for network names and host names.

For these reasons, we assume that the syntax of network names will be the same as the expanded syntax for host names permitted in Requirements for Internet Hosts. The new syntax expands the set of names to allow leading digits, so long as the resulting representations do not conflict with IP addresses in decimal octet form. For example, `3Com.COM` and `3M.COM` are now legal, although `26.0.0.73.COM` is not.

The price is that network names will get as complicated as host names. An administrator will be able to create network names in any domain under his control, and also create network number to name entries in `IN-ADDR.ARPA` domains under his control. Thus, the name for the ARPANET might become `NET.ARPA`, `ARPANET.ARPA` or `Arpa- network.MIL.`, depending on the preferences of the owner.

Mappings

The desired mappings, ranked by priority with most important first, are:

- Mapping a IP address or network number to a network name.

This mapping is for use in debugging tools and status displays of various sorts. The conversion from IP address to network number is well known for class A, B, and C IP addresses, and involves a simple mask operation. The needs of other classes are not yet defined and are ignored for the rest of this RFC.

- Mapping a network name to a network address.

This facility is of less obvious application, but a symmetrical mapping seems desirable.

- Mapping an organization to its network names and numbers.

This facility is useful because it may not always be possible to guess the local choice for network names, but the organization name is often well known.

- Similar mappings for subnets, even when nested.

The primary application is to be able to identify all of the subnets involved in a particular IP address. A secondary requirement is to retrieve address mask information.

Network Address Section of the Name Space

The network name syntax discussed above can provide domain names which will contain mappings from network names to various quantities, but we also need a section of the name space, organized by network and subnet number to hold the inverse mappings.

The choices include:

- The same network number slots already assigned and delegated in the IN-ADDR.ARPA section of the name space.

For example, `10.IN-ADDR.ARPA` for class A net 10, `2.128.IN-ADDR.ARPA` for class B net `128.2`, etc.

- Host-zero addresses in the IN-ADDR.ARPA tree. (A host field of all zero in an IP address is prohibited because of confusion related to broadcast addresses, et al.)

For example, `0.0.0.10.IN-ADDR.ARPA` for class A net 10, `0.0.2.128.IN-ADDR.arpa` for class B net `128.2`, etc. Like the first scheme, it uses in-place name space delegations to distribute control.

The main advantage of this scheme over the first is that it allows convenient names for subnets as well as networks. A secondary advantage is that it uses names which are not in use already, and hence it is possible to test whether an organization has entered this information in its domain database.

- Some new section of the name space.

While this option provides the most opportunities, it creates a need to delegate a whole new name space. Since the IP address space is so closely related to the network number space, most believe that the overhead of creating such a new space is overwhelming and would lead to the WKS syndrome. (As of February, 1989, approximately 400 sections of the IN-ADDR.ARPA tree are already delegated, usually at network boundaries.)

RFC-1101 DNS Encoding of Network Names and Other Types

Network Name Mappings

Specifics for Network Name Mappings

The proposed solution uses information stored at:

- Names in the IN-ADDR.ARPA tree that correspond to host-zero IP addresses. The same method is used for subnets in a nested fashion. For example, `0.0.0.10.IN-ADDR.ARPA`. for net 10.

Two types of information are stored here: PTR RRs which point to the network name in their data sections, and A RRs, which are present if the network (or subnet) is subnetted further. If a type A RR is present, then it has the address mask as its data. The general form is:

```
<reversed-host-zero-number>.IN-ADDR.ARPA. PTR <network-name>
<reversed-host-zero-number>.IN-ADDR.ARPA. A   <subnet-mask>
```

For example:

```
0.0.0.10.IN-ADDR.ARPA.   PTR     ARPANET.ARPA.
```

or

```
0.0.2.128.IN-ADDR.ARPA. PTR     cmu-net.cmu.edu.
                        A       255.255.255.0
```

In general, this information will be added to an existing master file for some IN-ADDR.ARPA domain for each network involved. Similar RRs can be used at host-zero subnet entries.

- Names which are network names.

The data stored here is PTR RRs pointing at the host-zero entries. The general form is:

```
<network-name> ptr <reversed-host-zero-number>.IN-ADDR.ARPA
```

For example:

```
ARPANET.ARPA.           PTR     0.0.0.10.IN-ADDR.ARPA.
```

or

```
isi-net.isi.edu.        PTR     0.0.9.128.IN-ADDR.ARPA.
```

In general, this information will be inserted in the master file for the domain name of the organization; this is a different file from that which holds the information below IN-ADDR.ARPA. Similar PTR RRs can be used at subnet names.

- Names corresponding to organizations.

The data here is one or more PTR RRs pointing at the IN-ADDR.ARPA names corresponding to host-zero entries for networks.

For example:

```
ISI.EDU.        PTR     0.0.9.128.IN-ADDR.ARPA.

MCC.COM.        PTR     0.167.5.192.IN-ADDR.ARPA.
                PTR     0.168.5.192.IN-ADDR.ARPA.
                PTR     0.169.5.192.IN-ADDR.ARPA.
                PTR     0.0.62.128.IN-ADDR.ARPA.
```

Network Name Mappings: A simple example

The ARPANET is a Class A network without subnets. The RRs which would be added, assuming the ARPANET.ARPA was selected as a network name, would be:

```
ARPA.                    PTR     0.0.0.10.IN-ADDR.ARPA.

ARPANET.ARPA.            PTR     0.0.0.10.IN-ADDR.ARPA.

0.0.0.10.IN-ADDR.ARPA.   PTR     ARPANET.ARPA.
```

The first RR states that the organization named ARPA owns net 10 (It might also own more network numbers, and these would be represented with an additional RR per net.) The second states that the network name ARPANET.ARPA. maps to net 10. The last states that net 10 is named ARPANET.ARPA.

Note that all of the usual host and corresponding IN-ADDR.ARPA entries would still be required.

Network Name Mappings: A Complicated, Subnetted Example

The ISI network is 128.9, a class B number. Suppose the ISI network was organized into two levels of subnet, with the first level using an additional 8 bits of address, and the second level using 4 bits, for address masks of x'FFFFFF00' and X'FFFFFFF0'.

Then the following RRs would be entered in ISI's master file for the ISI.EDU zone:

```
; Define network entry
isi-net.isi.edu.                PTR  0.0.9.128.IN-ADDR.ARPA.

; Define first level subnets
div1-subnet.isi.edu.            PTR  0.1.9.128.IN-ADDR.ARPA.
div2-subnet.isi.edu.            PTR  0.2.9.128.IN-ADDR.ARPA.

; Define second level subnets
inc-subsubnet.isi.edu.          PTR  16.2.9.128.IN-ADDR.ARPA.
```

in the 9.128.IN-ADDR.ARPA zone:

```
; Define network number and address mask
0.0.9.128.IN-ADDR.ARPA.         PTR  isi-net.isi.edu.
                                A    255.255.255.0  ;aka X'FFFFFF00'

; Define one of the first level subnet numbers and masks
0.1.9.128.IN-ADDR.ARPA.         PTR  div1-subnet.isi.edu.
                                A    255.255.255.240 ;aka X'FFFFFFF0'

; Define another first level subnet number and mask
0.2.9.128.IN-ADDR.ARPA.         PTR  div2-subnet.isi.edu.
                                A    255.255.255.240 ;aka X'FFFFFFF0'

; Define second level subnet number
16.2.9.128.IN-ADDR.ARPA.        PTR  inc-subsubnet.isi.edu.
```

This assumes that the ISI network is named isi-net.isi.edu., first level subnets are named div1-subnet.isi.edu. and div2- subnet.isi.edu., and a second level subnet is called inc-subsubnet.isi.edu. (In a real system as complicated as this there would be more first and second level subnets defined, but we have shown enough to illustrate the ideas.)

Procedure for Using an IP Address to Get Network Name

Depending on whether the IP address is class A, B, or C, mask off the high one, two, or three bytes, respectively. Reverse the octets, suffix IN-ADDR.ARPA, and do a PTR query.

For example, suppose the IP address is `10.0.0.51`.

1) Since this is a class A address, use a mask x'FF000000' and get 10.0.0.0.
2) Construct the name `0.0.0.10.IN-ADDR.ARPA`.
3) Do a PTR query. Get back

```
0.0.0.10.IN-ADDR.ARPA.   PTR     ARPANET.ARPA.
```

4) Conclude that the network name is "`ARPANET.ARPA`."

Suppose that the IP address is 128.9.2.17.

1) Since this is a class B address, use a mask of x'FFFF0000' and get `128.9.0.0`.
2) Construct the name `0.0.9.128.IN-ADDR.ARPA`.
3) Do a PTR query. Get back

```
0.0.9.128.IN-ADDR.ARPA.        PTR     isi-net.isi.edu
```

4) Conclude that the network name is "`isi-net.isi.edu`."

Procedure for Finding All Subnets Involved With an IP Address

This is a simple extension of the IP address to network name method. When the network entry is located, do a lookup for a possible A RR. If the A RR is found, look up the next level of subnet using the original IP address and the mask in the A RR. Repeat this procedure until no A RR is found.

For example, repeating the use of `128.9.2.17`.

1) As before construct a query for `0.0.9.128.IN-ADDR.ARPA`. Retrieve:

```
0.0.9.128.IN-ADDR.ARPA.  PTR    isi-net.isi.edu.
                         A      255.255.255.0
```

2) Since an A RR was found, repeat using mask from RR (`255.255.255.0`), constructing a query for `0.2.9.128.IN-ADDR.ARPA`. Retrieve:

```
0.2.9.128.IN-ADDR.ARPA.  PTR    div2-subnet.isi.edu.
                         A      255.255.255.240
```

3) Since another A RR was found, repeat using mask `255.255.255.240` (x'FFFFFFF0'). constructing a query for `16.2.9.128.IN-ADDR.ARPA`. Retrieve:

```
16.2.9.128.IN-ADDR.ARPA. PTR    inc-subsubnet.isi.edu.
```

4) Since no A RR is present at `16.2.9.128.IN-ADDR.ARPA`., there are no more subnet levels.

YP Issues and Discussion

The term "Yellow Pages" is used in almost as many ways as the term "domain", so it is useful to define what is meant herein by YP. The general problem to be solved is to create a method for creating mappings from one kind of identifier to another, often with an inverse capability. The traditional methods are to search or use a precomputed index of some kind.

Searching is impractical when the search is too large, and precomputed indexes are possible only when it is possible to specify search criteria in advance, and pay for the resources necessary to build the index. For example, it is impractical to search the entire domain tree to find a particular address RR, so we build the IN- ADDR.ARPA YP. Similarly, we could never build an Internet-wide index of "hosts with a load average of less than 2" in less time than it would take for the data to change, so indexes are a useless approach for that problem.

Such a precomputed index is what we mean by YP, and we regard the IN-ADDR.ARPA domain as the first instance of a YP in the DNS. Although a single, centrally-managed YP for well-known values such as TCP-port is desirable, we regard organization-specific YPs for, say, locally defined TCP ports as a natural extension, as are combinations of YPs using search lists to merge the two.

In examining Internet Numbers (RFC-997) and Assigned Numbers (RFC-1010), it is clear that there are several mappings which might be of value. For example:

```
<assigned-network-name> <==> <IP-address>
<autonomous-system-id>  <==> <number>
<protocol-id>           <==> <number>
<port-id>               <==> <number>
<ethernet-type>         <==> <number>
<public-data-net>       <==> <IP-address>
```

Following the IN-ADDR example, the YP takes the form of a domain tree organized to optimize retrieval by search key and distribution via normal DNS rules. The name used as a key must include:

1) A well known origin. For example, IN-ADDR.ARPA is the current IP-address to host name YP.

2) A "from" data type. This identifies the input type of the mapping. This is necessary because we may be mapping something as anonymous as a number to any number of mnemonics, etc.

3) A "to" data type. Since we assume several symmetrical mnemonic <==> number mappings, this is also necessary.

This ordering reflects the natural scoping of control, and hence the order of the components in a domain name. Thus domain names would be of the form:

```
<from-value>.<to-data-type>.<from-data-type>.<YP-origin>
```

To make this work, we need to define well-know strings for each of these metavariables, as well as encoding rules for converting a <`from-value`> into a domain name. We might define:

```
<YP-origin>     :=YP
<from-data-type>:=TCP-port | IN-ADDR | Number |
                  Assigned-network-number | Name
<to-data-type>  :=<from-data-type>
```

Note that "YP" is NOT a valid country code under ISO 3166 (although we may want to worry about the future), and the existence of a syntactically valid <to-data-type>.<from-data-type> pair does not imply that a meaningful mapping exists, or is even possible.

The encoding rules might be:

```
TCP-port        Six character alphanumeric

IN-ADDR         Reversed 4-octet decimal string

Number          decimal integer

Assigned-network-number
                Reversed 4-octet decimal string

Name            Domain name
```

RFC-1101 DNS Encoding of Network Names and Other Types

Specifics for YP Mappings

TCP-PORT
Assigned Networks
Operational Improvements

TCP-PORT

```
$origin Number.TCP-port.YP.

23              PTR     TELNET.TCP-port.Number.YP.
25              PTR     SMTP.TCP-port.Number.YP.

$origin TCP-port.Number.YP.


TELNET          PTR     23.Number.TCP-port.YP.
SMTP            PTR     25.Number.TCP-port.YP.
```

Thus the mapping between 23 and TELNET is represented by a pair of PTR RRs, one for each direction of the mapping.

Assigned Networks

Network numbers are assigned by the NIC and reported in "Internet Numbers" RFCs. To create a YP, the NIC would set up two domains:

```
Name.Assigned-network-number.YP and Assigned-network-number.YP
```

The first would contain entries of the form:

```
$origin Name.Assigned-network-number.YP.

0.0.0.4          PTR     SATNET.Assigned-network-number.Name.YP.
0.0.0.10         PTR     ARPANET.Assigned-network-number.Name.YP.
```

The second would contain entries of the form:

```
$origin Assigned-network-number.Name.YP.

SATNET.          PTR     0.0.0.4.Name.Assigned-network-number.YP.
ARPANET.         PTR     0.0.0.10.Name.Assigned-network-number.YP.
```

These YPs are not in conflict with the network name support described in the first half of this RFC since they map between ASSIGNED network names and numbers, not those allocated by the organizations themselves. That is, they document the NIC's decisions about allocating network numbers but do not automatically track any renaming performed by the new owners.

As a practical matter, we might want to create both of these domains to enable users on the Internet to experiment with centrally maintained support as well as the distributed version, or might want to implement only the allocated number to name mapping and request organizations to convert their allocated network names to the network names described in the distributed model.

Operational Improvements

We could imagine that all conversion routines using these YPs might be instructed to use "YP.<local-domain>" followed by "YP." as a search list. Thus, if the organization ISI.EDU wished to define locally meaningful TCP-PORT, it would define the domains:

```
<TCP-port.Number.YP.ISI.EDU> and <Number.TCP-port.YP.ISI.EDU>.
```

We could add another level of indirection in the YP lookup, defining the <to-data-type>.<from-data-type>.<YP-origin> nodes to point to the YP tree, rather than being the YP tree directly. This would enable entries of the form:

```
IN-ADDR.Netname.YP.    PTR     IN-ADDR.ARPA.
```

to splice in YPs from other origins or existing spaces.

Another possibility would be to shorten the RDATA section of the RRs which map back and forth by deleting the origin. This could be done either by allowing the domain name in the RDATA portion to not identify a real domain name, or by defining a new RR which used a simple text string rather than a domain name.

Thus, we might replace

```
$origin Assigned-network-number.Name.YP.

SATNET.          PTR     0.0.0.4.Name.Assigned-network-number.YP.
ARPANET.         PTR     0.0.0.10.Name.Assigned-network-number.YP.
```

with

```
$origin Assigned-network-number.Name.YP.

SATNET.          PTR     0.0.0.4.
ARPANET.         PTR     0.0.0.10.
```

or

```
$origin Assigned-network-number.Name.YP.

SATNET.          PTT     "0.0.0.4"
ARPANET.         PTT     "0.0.0.10"
```

where PTT is a new type whose RDATA section is a text string.

Acknowledgments

Drew Perkins, Mark Lottor, and Rob Austein contributed several of the ideas in this RFC. Numerous contributions, criticisms, and compromises were produced in the IETF Domain working group and the NAMEDROPPERS mailing list.

Braden, B., editor, "Requirements for Internet Hosts",
RFC in preparation.

ISO-3166, ISO, "Codes for the Representation of Names of Countries", 1981.

[RFC 882] Mockapetris, P., "Domain names - Concepts and Facilities", RFC 882, USC/Information Sciences Institute, November 1983.

Superseded by RFC 1034.

[RFC 883] Mockapetris, P.,"Domain names - Implementation and Specification", RFC 883, USC/Information Sciences Institute, November 1983.

Superceeded by RFC 1035.

[RFC 920] Postel, J. and J. Reynolds, "Domain Requirements", RFC 920, October 1984.

Explains the naming scheme for top level domains.

[RFC 952] Harrenstien, K., M. Stahl, and E. Feinler, "DoD Internet Host Table Specification", RFC 952, SRI, October 1985.

Specifies the format of HOSTS.TXT, the host/address table replaced by the DNS

[RFC 973] Mockapetris, P., "Domain System Changes and Observations", RFC 973, USC/Information Sciences Institute, January 1986.

Describes changes to RFCs 882 and 883 and reasons for them.

[RFC 974] Partridge, C., "Mail routing and the domain system", RFC 974, CSNET CIC BBN Labs, January 1986.

Describes the transition from HOSTS.TXT based mail addressing to the more powerful MX system used with the domain system.

[RFC 997] Reynolds, J., and J. Postel, "Internet Numbers", RFC 997, USC/Information Sciences Institute, March 1987

Contains network numbers, autonomous system numbers, etc.

[RFC 1010] Reynolds, J., and J. Postel, "Assigned Numbers", RFC 1010, USC/Information Sciences Institute, May 1987

Contains socket numbers and mnemonics for host names, operating systems, etc.

[RFC 1034] Mockapetris, P., "Domain names - Concepts and Facilities", RFC 1034, USC/Information Sciences Institute, November 1987.

Introduction/overview of the DNS.

[RFC 1035] Mockapetris, P., "Domain names - Implementation and Specification", RFC 1035, USC/Information Sciences Institute, November 1987.

DNS implementation instructions.

RFC-1101 DNS Encoding of Network Names and Other Types

Author's Address:

Paul Mockapetris
USC/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292

Phone: (213) 822-1511
Email: PVM@ISI.EDU

RFC-1112 Host Extensions for IP Multicasting

Status of this Memo

This memo specifies the extensions required of a host implementation of the Internet Protocol (IP) to support multicasting. It is the recommended standard for IP multicasting in the Internet. Distribution of this memo is unlimited.

RFC-1112 Host Extensions for IP Multicasting

Introduction

IP multicasting is the transmission of an IP datagram to a "host group", a set of zero or more hosts identified by a single IP destination address. A multicast datagram is delivered to all members of its destination host group with the same "best-efforts" reliability as regular unicast IP datagrams, i.e., the datagram is not guaranteed to arrive intact at all members of the destination group or in the same order relative to other datagrams.

The membership of a host group is dynamic; that is, hosts may join and leave groups at any time. There is no restriction on the location or number of members in a host group. A host may be a member of more than one group at a time. A host need not be a member of a group to send datagrams to it.

A host group may be permanent or transient. A permanent group has a well-known, administratively assigned IP address. It is the address, not the membership of the group, that is permanent; at any time a permanent group may have any number of members, even zero. Those IP multicast addresses that are not reserved for permanent groups are available for dynamic assignment to transient groups which exist only as long as they have members.

Internetwork forwarding of IP multicast datagrams is handled by "multicast routers" which may be co-resident with, or separate from, internet gateways. A host transmits an IP multicast datagram as a local network multicast which reaches all immediately-neighboring members of the destination host group. If the datagram has an IP time-to-live greater than 1, the multicast router(s) attached to the local network take responsibility for forwarding it towards all other networks that have members of the destination group. On those other member networks that are reachable within the IP time-to-live, an attached multicast router completes delivery by transmitting the datagram as a local multicast.

This memo specifies the extensions required of a host IP implementation to support IP multicasting, where a "host" is any internet host or gateway other than those acting as multicast routers. The algorithms and protocols used within and between multicast routers are transparent to hosts and will be specified in separate documents. This memo also does not specify how local network multicasting is accomplished for all types of network, although it does specify the required service interface to an arbitrary local network and gives an Ethernet specification as an example. Specifications for other types of network will be the subject of future memos.

Levels of Conformance

There are three levels of conformance to this specification:

Level 0: no support for IP multicasting

There is, at this time, no requirement that all IP implementations support IP multicasting. Level 0 hosts will, in general, be unaffected by multicast activity. The only exception arises on some types of local network, where the presence of level 1 or 2 hosts may cause misdelivery of multicast IP datagrams to level 0 hosts. Such datagrams can easily be identified by the presence of a class D IP address in their destination address field; they should be quietly discarded by hosts that do not support IP multicasting. Class D addresses are described in Host Group Address.

Level 1: support for sending but not receiving multicast IP datagrams

Level 1 allows a host to partake of some multicast-based services, such as resource location or status reporting, but it does not allow a host to join any host groups. An IP implementation may be upgraded from level 0 to level 1 very easily and with little new code. Host Group Addresses, Model of a Host IP Implementation, and Sending Multicast IP Datagrams are relevant for level 1 implementations.

Level 2: full support for IP multicasting

Level 2 allows a host to join and leave host groups, as well as send IP datagrams to host groups. It requires implementation of the Internet Group Management Protocol (IGMP) and extension of the IP and local network service interfaces within the host. All of the following sections of this memo are applicable to level 2 implementations.

Host Group Addresses

Host groups are identified by class D IP addresses, i.e., those with "1110" as their high-order four bits. Class E IP addresses, i.e., those with "1111" as their high-order four bits, are reserved for future addressing modes.

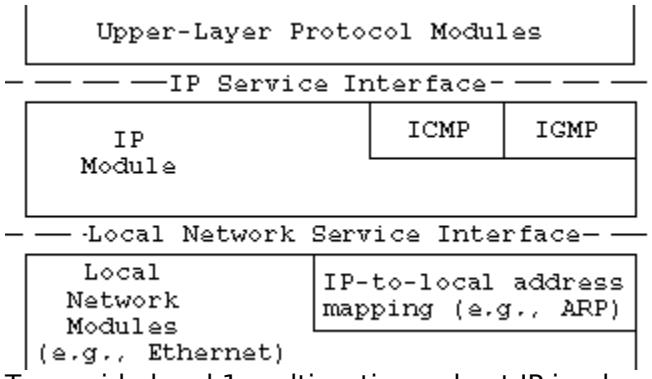
In Internet standard "dotted decimal" notation, host group addresses range from 224.0.0.0 to 239.255.255.255. The address 224.0.0.0 is guaranteed not to be assigned to any group, and 224.0.0.1 is assigned to the permanent group of all IP hosts (including gateways). This is used to address all multicast hosts on the directly connected network. There is no multicast address (or any other IP address) for all hosts on the total Internet. The addresses of other well-known, permanent groups are to be published in "Assigned Numbers".

The Appendix contains some background discussion of several issues related to host group addresses.

RFC-1112 Host Extensions for IP Multicasting

Model of a Host IP Implementation

The multicast extensions to a host IP implementation are specified in terms of the layered model illustrated below. In this model, ICMP and (for level 2 hosts) IGMP are considered to be implemented within the IP module, and the mapping of IP addresses to local network addresses is considered to be the responsibility of local network modules. This model is for expository purposes only, and should not be construed as constraining an actual implementation.



To provide level 1 multicasting, a host IP implementation must support the transmission of multicast IP datagrams. To provide level 2 multicasting, a host must also support the reception of multicast IP datagrams. Each of these two new services is described in a separate section, below. For each service, extensions are specified for the IP service interface, the IP module, the local network service interface, and an Ethernet local network module. Extensions to local network modules other than Ethernet are mentioned briefly, but are not specified in detail.

Extensions to the IP Service Interface

Multicast IP datagrams are sent using the same "Send IP" operation used to send unicast IP datagrams; an upper-layer protocol module merely specifies an IP host group address, rather than an individual IP address, as the destination. However, a number of extensions may be necessary or desirable.

First, the service interface should provide a way for the upper-layer protocol to specify the IP time-to-live of an outgoing multicast datagram, if such a capability does not already exist. If the upper-layer protocol chooses not to specify a time-to-live, it should default to 1 for all multicast IP datagrams, so that an explicit choice is required to multicast beyond a single network.

Second, for hosts that may be attached to more than one network, the service interface should provide a way for the upper-layer protocol to identify which network interface is be used for the multicast transmission. Only one interface is used for the initial transmission; multicast routers are responsible for forwarding to any other networks, if necessary. If the upper-layer protocol chooses not to identify an outgoing interface, a default interface should be used, preferably under the control of system management.

Third (level 2 implementations only), for the case in which the host is itself a member of a group to which a datagram is being sent, the service interface should provide a way for the upper-layer protocol to inhibit local delivery of the datagram; by default, a copy of the datagram is looped back. This is a performance optimization for upper-layer protocols that restrict the membership of a group to one process per host (such as a routing protocol), or that handle loopback of group communication at a higher layer (such as a multicast transport protocol).

Extensions to the IP Module

To support the sending of multicast IP datagrams, the IP module must be extended to recognize IP host group addresses when routing outgoing datagrams. Most IP implementations include the following logic:

```
if IP-destination is on the same local network,
        send datagram locally to IP-destination
else
        send datagram locally to GatewayTo( IP-destination )
```

To allow multicast transmissions, the routing logic must be changed to:

```
if IP-destination is on the same local network
or IP-destination is a host group,
        send datagram locally to IP-destination
else
        send datagram locally to GatewayTo( IP-destination )
```

If the sending host is itself a member of the destination group on the outgoing interface, a copy of the outgoing datagram must be looped-back for local delivery, unless inhibited by the sender. (Level 2 implementations only.)

The IP source address of the outgoing datagram must be one of the individual addresses corresponding to the outgoing interface.

A host group address must never be placed in the source address field or anywhere in a source route or record route option of an outgoing IP datagram.

Extensions to the Local Network Service Interface

No change to the local network service interface is required to support the sending of multicast IP datagrams. The IP module merely specifies an IP host group destination, rather than an individual IP destination, when it invokes the existing "Send Local" operation.

Extensions to an Ethernet Local Network Module

The Ethernet directly supports the sending of local multicast packets by allowing multicast addresses in the destination field of Ethernet packets. All that is needed to support the sending of multicast IP datagrams is a procedure for mapping IP host group addresses to Ethernet multicast addresses.

An IP host group address is mapped to an Ethernet multicast address by placing the low-order 23-bits of the IP address into the low-order 23 bits of the Ethernet multicast address 01-00-5E-00-00-00 (hex). Because there are 28 significant bits in an IP host group address, more than one host group address may map to the same Ethernet multicast address.

Extensions to Local Network Modules other than Ethernet

Other networks that directly support multicasting, such as rings or buses conforming to the IEEE 802.2 standard, may be handled the same way as Ethernet for the purpose of sending multicast IP datagrams. For a network that supports broadcast but not multicast, such as the Experimental Ethernet, all IP host group addresses may be mapped to a single local broadcast address (at the cost of increased overhead on all local hosts). For a point-to-point link joining two hosts (or a host and a multicast router), multicasts should be transmitted exactly like unicasts. For a store-and-forward network like the ARPANET or a public X.25 network, all IP host group addresses might be mapped to the well-known local address of an IP multicast router; a router on such a network would take responsibility for completing multicast delivery within the network as well as among networks.

Extensions to the IP Service Interface

Incoming multicast IP datagrams are received by upper-layer protocol modules using the same "Receive IP" operation as normal, unicast datagrams. Selection of a destination upper-layer protocol is based on the protocol field in the IP header, regardless of the destination IP address. However, before any datagrams destined to a particular group can be received, an upper-layer protocol must ask the IP module to join that group. Thus, the IP service interface must be extended to provide two new operations:

JoinHostGroup (group-address, interface)

LeaveHostGroup (group-address, interface)

The JoinHostGroup operation requests that this host become a member of the host group identified by "group-address" on the given network interface. The LeaveGroup operation requests that this host give up its membership in the host group identified by "group-address" on the given network interface. The interface argument may be omitted on hosts that support only one interface. For hosts that may be attached to more than one network, the upper-layer protocol may choose to leave the interface unspecified, in which case the request will apply to the default interface for sending multicast datagrams (see section 6.1).

It is permissible to join the same group on more than one interface, in which case duplicate multicast datagrams may be received. It is also permissible for more than one upper-layer protocol to request membership in the same group.

Both operations should return immediately (i.e., they are non- blocking operations), indicating success or failure. Either operation may fail due to an invalid group address or interface identifier. JoinHostGroup may fail due to lack of local resources.

LeaveHostGroup may fail because the host does not belong to the given group on the given interface. LeaveHostGroup may succeed, but the membership persist, if more than one upper-layer protocol has requested membership in the same group.

RFC-1112 Receiving Multicast IP Datagrams

Extensions to the IP Module

To support the reception of multicast IP datagrams, the IP module must be extended to maintain a list of host group memberships associated with each network interface. An incoming datagram destined to one of those groups is processed exactly the same way as datagrams destined to one of the host's individual addresses.

Incoming datagrams destined to groups to which the host does not belong are discarded without generating any error report or log entry. On hosts with more than one network interface, if a datagram arrives via one interface, destined for a group to which the host belongs only on a different interface, the datagram is quietly discarded. (These cases should occur only as a result of inadequate multicast address filtering in a local network module.)

An incoming datagram is not rejected for having an IP time-to-live of 1 (i.e., the time-to-live should not automatically be decremented on arriving datagrams that are not being forwarded). An incoming datagram with an IP host group address in its source address field is quietly discarded. An ICMP error message (Destination Unreachable, Time Exceeded, Parameter Problem, Source Quench, or Redirect) is never generated in response to a datagram destined to an IP host group.

The list of host group memberships is updated in response to JoinHostGroup and LeaveHostGroup requests from upper-layer protocols. Each membership should have an associated reference count or similar mechanism to handle multiple requests to join and leave the same group. On the first request to join and the last request to leave a group on a given interface, the local network module for that interface is notified, so that it may update its multicast reception filter.

The IP module must also be extended to implement the IGMP protocol. IGMP is used to keep neighboring multicast routers informed of the host group memberships present on a particular local network. To support IGMP, every level 2 host must join the "all-hosts" group (address 224.0.0.1) on each network interface at initialization time and must remain a member for as long as the host is active.

(Datagrams addressed to the all-hosts group are recognized as a special case by the multicast routers and are never forwarded beyond a single network, regardless of their time-to-live. Thus, the all- hosts address may not be used as an internet-wide broadcast address. For the purpose of IGMP, membership in the all-hosts group is really necessary only while the host belongs to at least one other group. However, it is specified that the host shall remain a member of the all-hosts group at all times because (1) it is simpler, (2) the frequency of reception of unnecessary IGMP queries should be low enough that overhead is negligible, and (3) the all-hosts address may serve other routing-oriented purposes, such as advertising the presence of gateways or resolving local addresses.)

Extensions to the Local Network Service Interface

Incoming local network multicast packets are delivered to the IP module using the same "Receive Local" operation as local network unicast packets. To allow the IP module to tell the local network module which multicast packets to accept, the local network service interface is extended to provide two new operations:

JoinLocalGroup (group-address)

LeaveLocalGroup (group-address)

where "group-address" is an IP host group address. The JoinLocalGroup operation requests the local network module to accept and deliver up subsequently arriving packets destined to the given IP host group address. The LeaveLocalGroup operation requests the local network module to stop delivering up packets destined to the given IP host group address. The local network module is expected to map the IP host group addresses to local network addresses as required to update its multicast reception filter. Any local network module is free to ignore LeaveLocalGroup requests, and may deliver up packets destined to more addresses than just those specified in JoinLocalGroup requests, if it is unable to filter incoming packets adequately.

The local network module must not deliver up any multicast packets that were transmitted from that module; loopback of multicasts is handled at the IP layer or higher.

Extensions to an Ethernet Local Network Module

To support the reception of multicast IP datagrams, an Ethernet module must be able to receive packets addressed to the Ethernet multicast addresses that correspond to the host's IP host group addresses. It is highly desirable to take advantage of any address filtering capabilities that the Ethernet hardware interface may have, so that the host receives only those packets that are destined to it.

Unfortunately, many current Ethernet interfaces have a small limit on the number of addresses that the hardware can be configured to recognize. Nevertheless, an implementation must be capable of listening on an arbitrary number of Ethernet multicast addresses, which may mean "opening up" the address filter to accept all multicast packets during those periods when the number of addresses exceeds the limit of the filter.

For interfaces with inadequate hardware address filtering, it may be desirable (for performance reasons) to perform Ethernet address filtering within the software of the Ethernet module. This is not mandatory, however, because the IP module performs its own filtering based on IP destination addresses.

Extensions to Local Network Modules other than Ethernet

Other multicast networks, such as IEEE 802.2 networks, can be handled the same way as Ethernet for the purpose of receiving multicast IP datagrams. For pure broadcast networks, such as the Experimental Ethernet, all incoming broadcast packets can be accepted and passed to the IP module for IP-level filtering. On point-to-point or store-and-forward networks, multicast IP datagrams will arrive as local network unicasts, so no change to the local network module should be necessary.

Appendix

Host Group Address Issues

This appendix is not part of the IP multicasting specification, but provides background discussion of several issues related to IP host group addresses.

Group Address Binding

The binding of IP host group addresses to physical hosts may be considered a generalization of the binding of IP unicast addresses. An IP unicast address is statically bound to a single local network interface on a single IP network. An IP host group address is dynamically bound to a set of local network interfaces on a set of IP networks.

It is important to understand that an IP host group address is NOT bound to a set of IP unicast addresses. The multicast routers do not need to maintain a list of individual members of each host group. For example, a multicast router attached to an Ethernet need associate only a single Ethernet multicast address with each host group having local members, rather than a list of the members' individual IP or Ethernet addresses.

Allocation of Transient Host Group Addresses

This memo does not specify how transient group address are allocated. It is anticipated that different portions of the IP transient host group address space will be allocated using different techniques. For example, there may be a number of servers that can be contacted to acquire a new transient group address. Some higher-level protocols (such as VMTP, specified in RFC-1045) may generate higher- level transient "process group" or "entity group" addresses which are then algorithmically mapped to a subset of the IP transient host group addresses, similarly to the way that IP host group addresses are mapped to Ethernet multicast addresses. A portion of the IP group address space may be set aside for random allocation by applications that can tolerate occasional collisions with other multicast users, perhaps generating new addresses until a suitably "quiet" one is found.

In general, a host cannot assume that datagrams sent to any host group address will reach only the intended hosts, or that datagrams received as a member of a transient host group are intended for the recipient. Misdelivery must be detected at a level above IP, using higher-level identifiers or authentication tokens. Information transmitted to a host group address should be encrypted or governed by administrative routing controls if the sender is concerned about unwanted listeners.

RFC-1147 (FYI 2)

FYI on a
Network Management Tool Catalog:

Tools for Monitoring and Debugging TCP/IP Internets
and
Interconnected Devices

R. Stine Editor
April 1990

Status of this Memo

The goal of this FYI memo is to provide practical information to site administrators and network managers. This memo provides information for the Internet community. It does not specify any standard. It is not a statement of IAB policy or recommendations. Comments, critiques, and new or updated tool descriptions are welcome, and should be sent to Robert Stine, at stine@sparta.com, or to the NOCTools working group, at noctools@merit.edu.

Distribution of this memo is unlimited. Security issues are discussed in the seciton on Internet Security.

RFC-1147 Network Tools Directory

Introduction

This catalog contains descriptions of several tools available to assist network managers in debugging and maintaining TCP/IP internets and interconnected communications resources. Entries in the catalog tell what a tool does, how it works, and how it can be obtained.

The NOCTools Working Group of the Internet Engineering Task Force (IETF) compiled this catalog in 1989. Future editions will be produced as IETF members become aware of tools that should be included, and of deficiencies or inaccuracies. Developing an edition oriented to the OSI protocol suite is also contemplated.

The tools described in this catalog are in no way endorsed by the IETF. For the most part, we have neither evaluated the tools in this catalog, nor validated their descriptions. Most of the descriptions of commercial tools have been provided by vendors. Caveat Emptor.

Purpose

The practice of re-inventing the wheel seems endemic to the field of data communications. The primary goal of this document is to fight that tendency in a small but useful way. By listing the capabilities of some of the available network management tools, we hope to pool and share knowledge and experience. Another goal of this catalog is to show those new in the field what can be done to manage internet sites. A network management tutorial at the end of the document is of further assistance in this area. Finally, by omission, this catalog points out the network management tools that are needed, but do not yet exist.

There are other sources of information on available network management tools. Both the DDN Protocol Implementation and Vendors Guide and the DATAPRO series on data communications and LANs are particularly comprehensive and informative. The DDN Protocol Implementation and Vendors Guide addresses a wide range of internet management topics, including evaluations of protocol implementations and network analyzers.(Instructions for obtaining the DDN Protocol Guide are given in Section 7 of the appendix.) The DATAPRO volumes, though expensive (check your local university or technical libraries!), are good surveys of available commercial products for network management. DATAPRO also includes tutorials, market analyses, product evaluations, and predictions on technology trends.

Scope

The tools described in this document are used for managing the network resources, LANs, and devices that are commonly interconnected by TCP/IP internets. This document is not, however, a "how to" manual on network management. While it includes a tutorial, the coverage is much too brief and general to serve as a sole source: a great deal of further study is required of aspiring network managers. Neither is this catalog is an operations manual for particular tools. Each individual tool entry is brief, and emphasizes the uses to which a tool can be put. A tool's documentation, which in some cases runs to hundreds of pages, should be consulted for assistance in its installation and operation.

Overview

Section 1 describes the purpose, scope, and organization of this catalog.

Section 2 lists and explains the standard keywords used in the tool descriptions. The keywords can be used as a subject index into the catalog.

Section 3, the main body of the catalog, contains the entries describing network management tools. The tool entries in Section 3 are presented in alphabetical order, by tool name. The tool descriptions all follow a standard format, described in the introduction to Section 3.

Following the catalog, there is an appendix that contains a tutorial on the goals and practice of network management.

Acknowledgements

The compilation and editing of this catalog was sponsored by the Defense Communications Engineering Center (DCEC), contract DCA100-89-C-0001. The effort grew out of an initial task to survey current internet management tools. The cata- log is largely, however, the result of volunteer labor on the part of the NOCTools Working Group, the User Services Working Group, and many others. Without these volunteer contributions, the catalog would not exist. The support from the Internet community for this endeavor has been extremely gratifying.

Several individuals made especially notable contributions. Mike Patton, Paul Holbrook, Mark Fedor and Gary Malkin were particularly helpful in composition and editorial review, while Dave Crocker provided essential guidance and encouragement. Bob Enger was active from the first with the gut work of chairing the Working Group and building the catalog. Phill Gross helped to christen the NOCTools Working Group, to define its scope and goals, and to establish its role in the IETF. Mike Little contributed the formative idea of enhancing and publicizing the management tool survey through IETF participation.

Responsibility for any deficiencies and errors remains, of course, with the editor.

Keywords

This catalog uses "keywords" for terse characterizations of the tools. Keywords are abbreviated attributes of a tool or its use. To allow cross-comparison of tools, uniform keyword definitions have been developed, and are given below. Following the definitions, there is an index of catalog entries by keyword.

Keyword Definitions

The keywords are always listed in a prefined order, sorted first by the general category into which they fall, and then alphabetically. The categories that have been defined for management tool keywords are:

- the general management area to which a tool relates or a tool's functional role;
- the network resources or components that are managed;
- the mechanisms or methods a tool uses to perform its functions;
- the operating system and hardware environment of a tool; and
- the characteristics of a tool as a hardware product or software release.

RFC-1147 Network Tools Directory - Keywords

General Management Area or Functional Role Keywords

Alarm

a reporting/logging tool that can trigger on specific events within a network.

Analyzer

a traffic monitor that reconstructs and interprets protocol messages that span several packets.

Benchmark

a tool used to evaluate the performance of network com ponents.

Control

a tool that can change the state or status of a remote network resource.

Debugger

a tool that by generating arbitrary packets and moni toring traffic, can drive a remote network component to various states and record its responses.

Generator

a traffic generation tool.

Manager

a distributed network management system or system com ponent.

Map

a tool that can discover and report a system's topology or configuration.

Reference

a tool for documenting MIB structure or system configuration.

Routing

a packet route discovery tool.

Security

a tool for analyzing or reducing threats to security.

Status

a tool that remotely tracks the status of network components.

Traffic

a tool that monitors packet flow.

RFC-1147 Network Tools Directory - Keywords

Network Resources or Managed Components Keywords

Bridge

a tool for controlling or monitoring LAN bridges.

CHAOS

a tool for controlling or monitoring implementations of the CHAOS protocol suite or network components that use it.

DECnet

a tool for controlling or monitoring implementations of the DECnet protocol suite or network components that use it.

DNS

a Domain Name System debugging tool.

Ethernet

a tool for controlling or monitoring network components on ethernet LANs.

FDDI

a tool for controlling or monitoring network components on FDDI LANs or WANs.

IP

a tool for controlling or monitoring implementations of the TCP/IP protocol suite or network components that use it.

OSI

a tool for controlling or monitoring implementations of the OSI protocol suite or network components that use it.

NFS

a Network File System debugging tool.

Ring

a tool for controlling or monitoring network components on Token Ring LANs.

SMTP

an SMTP debugging tool.

Star

a tool for controlling or monitoring network components on StarLANs.

Mechanism Keywords

Curses

a tool that uses the "curses" tty interface package.

Eavesdrop

a tool that silently monitors communications media (e.g., by putting an ethernet interface into "promiscuous" mode).

NMS

the tool is a component of or queries a Network Manage ment System.

Ping

a tool that sends packet probes such as ICMP echo messages; to help distinguish tools, we do not consider

NMS queries or protocol spoofing (see below) as probes.

Proprietary

a distributed tool that uses proprietary communications techniques to link its components.

SNMP

a network management system or component based on SNMP, the Simple Network Management Protocol.

Spoof

a tool that tests operation of remote protocol modules by peer-level message exchange.

X

a tool that uses X-Windows.

Operating Environment Keywords

DOS

a tool that runs under MS-DOS.

HP

a tool that runs on Hewlett-Packard systems.

Macintosh

a tool that runs on Macintosh personal computers.

Standalone

an integrated hardware/software tool that requires only a network interface for operation.

UNIX

a tool that runs under 4.xBSD UNIX or related OS.

VMS

a tool that runs under DEC's VMS operating system.

Acquisition Keywords

Free

a tool is available at no charge, though other restrictions may apply (tools that are part of an OS distribution but not otherwise available are not listed as "free").

Library

a tool packaged with either an Application Programming Interface (API) or object-level subroutines that may be loaded with programs.

Sourcelib

a collection of source code (subroutines) upon which developers may construct other tools.

Tools Indexed by Keywords

Following is an index of catalog entries sorted by keyword. This index can be used to locate the tools with a particular attribute: tools are listed under each keyword that characterizes them. The keywords and the subordinate lists of tools under them are in alphabetical order.

In the interest of brevity, some liberties have been taken with tool names. Capitalization of the names is as specified by the tool developers or distributers. Note that parenthetical roman numerals following a tool's name are not actually part of the name. The use of roman numerals to differentiate tools with the same name is explained in the introduction of Section 3.

RFC-1147 Network Tools Directory - Groups

alarm

CMIP Library
EtherMeter
LanProbe
LANWatch
NETMON (III)
osilog
SERAG
SpiderMonitor
sma
Snmp Libraries
snmptrapd
Unisys NCC
WIN/MGT Station
xnetmon (I)
XNETMON (II)

RFC-1147 Network Tools Directory - Groups

analyzer

LANWatch Sniffer SpiderMonitor

RFC-1147 Network Tools Directory - Groups

benchmark

hammer
nhfsstone
SPIMS
spray
TTCP
Unisys NCC

RFC-1147 Network Tools Directory - Groups

bridge

ConnectVIEW	NMC	snmpd (I)
decaddrs	proxyd	Snmp Libraries

RFC-1147 Network Tools Directory - Groups

CHAOS

LANWatch map

RFC-1147 Network Tools Directory - Groups

control

CMIP Library
ConnectVIEW
NETMON (III)
NMC
proxyd
snmpset
Snmp Libraries
TokenVIEW
Unisys NCC
WIN/MGT Station
XNETMON (II)

RFC-1147 Network Tools Directory - Groups

curses

Internet Rover	nfswatch	snmpperfmon
net_monitor	osimon	

RFC-1147 Network Tools Directory - Groups

debugger

SPIMS

RFC-1147 Network Tools Directory - Groups

DECnet

decaddrs
LANWatch
NETMON (III)
net_monitor
NMC
Sniffer
Snmp Libraries
SpiderMonitor
XNETMON (II)
xnetperfmon

RFC-1147 Network Tools Directory - Groups

DNS

DiG
LANWatch
netmon (I)
nslookup

RFC-1147 Network Tools Directory - Groups

DOS

Comp. Security Checklist
ConnectVIEW
hammer
hopcheck
LAN Patrol
LANWatch
netmon (I)
NETMON (III)
netwatch
OverVIEW
ping
Snmp Libraries
snmpd (II)
TokenVIEW
XNETMON (II)
xnetperfmon

RFC-1147 Network Tools Directory - Groups

eavesdrop

ENTM
etherfind
EtherView
LAN Patrol
LanProbe
LANWatch
NETMON (II)
netwatch
nfswatch
NNStat
OSITRACE
Sniffer
SpiderMonitor
Tcplogger
TRPT

RFC-1147 Network Tools Directory - Groups

ethernet

arp
ConnectVIEW
ENTM
etherfind
etherhostprobe
EtherMeter
EtherView
LAN Patrol
LanProbe
LANWatch
map
NETMON (III)
netwatch
Network Integrator
NMC
NNStat
proxyd
SERAG
Sniffer
Snmp Libraries
snmpd (II)
SpiderMonitor
tcpdump
Unisys NCC
WIN/MGT Station
XNETMON (II)
xnetperfmon

RFC-1147 Network Tools Directory - Groups

FDDI

Unisys NCC

RFC-1147 Network Tools Directory - Groups

free

arp
CMIP Library
CMU SNMP
DiG
ENTM
etherhostprobe
hammer
hopcheck
HyperMIB
Internet Rover
map
netmon (I)
NETMON (II)
netstat
netwatch
net_monitor
nfswatch
nhfsstone
NNStat
NPRV
nslookup
osilog
osimic
osimon
OSITRACE
ping
query
sma
SNMP Kit
tcpdump
tcplogger
traceroute
TRPT
TTCP

RFC-1147 Network Tools Directory - Groups

generator

hammer	Sniffer	TTCP
nhfsstone	SpiderMonitor	Unisys NCC
ping	spray	

RFC-1147 Network Tools Directory - Groups

HP

xup

RFC-1147 Network Tools Directory - Groups

IP

arp
CMU SNMP
Dual Manager
ENTM
etherfind
etherhostprobe
EtherView
getone
hammer
hopcheck
Internet Rover
LANWatch
map
Netlabs CMOT Agent
Netlabs SNMP Agent
netmon (I)
NETMON (II)
NETMON (III)
netstat
netwatch
net_monitor
nfswatch
NMC
NNStat
NPRV
OverVIEW
ping
proxyd
query
SERAG
Sniffer
SNMP Kit
Snmp Libraries
snmpack
snmpd (I)
snmpd (II)
snmplookup
snmpperfmon
snmppoll
snmpquery
snmproute
snmpset
snmpsrc
snmpstat
snmptrapd
snmpwatch
snmpxbar
snmpxconn
snmpxmon
snmpxperf
snmpxperfmon
snmpxrtmetric
SpiderMonitor
SPIMS
spray
Tcpdump
Tcplogger
Traceroute
TRPT
TTCP
Unisys NCC
WIN/MGT Station
xnetmon (I)
XNETMON (II)
xnetperfmon

RFC-1147 Network Tools Directory - Groups

library

CMIP Library
Dual Manager
LANWatch
proxyd
WIN/MGT Station

RFC-1147 Network Tools Directory - Groups

Macintosh

HyperMIB

RFC-1147 Network Tools Directory - Groups

manager

CMIP Library
CMU SNMP
ConnectVIEW
decaddrs
Dual Manager
getone
LanProbe
map
Netlabs CMOT Agent
Netlabs SNMP Agent
NETMON (III)
NMC
NNStat
osilog
osimic
osimon
OverVIEW
sma
SNMP Kit
Snmp Libraries
snmpask
snmpd (I)
snmpd (II)
snmplookup
snmpperfmon
snmppoll
snmpquery
snmproute
snmpsrc
snmpset
snmpstat
snmptrapd
snmpwatch
snmpxbar
snmpxconn
snmpxmon
snmpxperf
snmpxperfmon
snmpxrtmetric
TokenVIEW
Unisys NCC
WIN/MGT Station
xnetmon (I)
XNETMON (II)
xnetperfmon

RFC-1147 Network Tools Directory - Groups

map

decaddrs	NETMON (III)	snmpxmon
etherhostprobe	Network Integrator	Unisys NCC
EtherMeter	NPRV	xnetmon (I)
LanProbe	Snmp Libraries	XNETMON (II)
map	snmpxconn	

RFC-1147 Network Tools Directory - Groups

NFS

etherfind	nfswatch	Sniffer
EtherView	nhfsstone	tcpdump

RFC-1147 Network Tools Directory - Groups

NMS

CMU SNMP
ConnectVIEW
decaddrs
Dual Manager
EtherMeter
getone
LanProbe
map
Netlabs CMOT Agent
Netlabs SNMP Agent
NETMON (III)
NMC
NNStat
OverVIEW
proxyd
SERAG
SNMP Kit
Snmp Libraries
snmpask
snmpd (I)
snmpd (II)
snmplookup
snmpperfmon
snmppoll
snmpquery
snmproute
snmpset
snmpsrc
snmpstat
snmptrapd
snmpwatch
snmpxbar
snmpxconn
snmpxmon
snmpxperf
snmpxperfmon
snmpxrtmetric
TokenVIEW
Unisys NCC
WIN/MGT Station
xnetmon (I)
XNETMON (II)
xnetperfmon

RFC-1147 Network Tools Directory - Groups

OSI

CMIP Library
Dual Manager
LANWatch
Netlabs CMOT Agent
NETMON (III)
osilog
osimic
osimon
OSITRACE
sma
Sniffer
Snmp Libraries
SpiderMonitor
SPIMS
XNETMON (II)
xnetperfmon

RFC-1147 Network Tools Directory - Groups

ping

etherhostprobe	net_monitor	TTCP
hopcheck	NPRV	Unisys NCC
Internet Rover	ping	xup
map	spray	
netmon (I)	traceroute	

RFC-1147 Network Tools Directory - Groups

proprietary

ConnectVIEW
EtherMeter
LanProbe
SERAG
TokenVIEW

RFC-1147 Network Tools Directory - Groups

reference

HyperMIB Unisys NCC

RFC-1147 Network Tools Directory - Groups

ring

ConnectVIEW	netwatch	snmpd (II)
LANWatch	proxyd	TokenVIEW
map	Sniffer	XNETMON (II)
NETMON (III)	Snmp Libraries	xnetperfmon

RFC-1147 Network Tools Directory - Groups

routing

arp
ConnectVIEW
decaddrs
etherhostprobe
getone
hopcheck
NETMON (III)
netstat
net_monitor
NMC
NPRV
query
Snmp Libraries
snmproute
snmpsrc
snmpxrtmetric
traceroute
WIN/MGT Station
XNETMON (II)

RFC-1147 Network Tools Directory - Groups

security

Comp. Security Checklist	Dual Manager	SERAG
ConnectVIEW	LAN Patrol	XNETMON (II)

RFC-1147 Network Tools Directory - Groups

SMTP

Internet Rover	mconnect
LANWatch	Sniffer

RFC-1147 Network Tools Directory - Groups

SNMP

CMU SNMP
decaddrs
Dual Manager
getone
map
Netlabs SNMP Agent
NETMON (III)
NMC
OverVIEW
proxyd
SNMP Kit
Snmp Libraries
snmpask
snmpd (I)
snmpd (II)
snmplookup
snmpperfmon
snmppoll
snmpquery
snmproute
snmpset
snmpsrc
snmpstat
snmptrapd
snmpwatch
snmpxbar
snmpxconn
snmpxmon
snmpxperf
snmpxperfmon
snmpxrtmetric
Unisys NCC
WIN/MGT Station
xnetmon (I)
XNETMON (II)
xnetperfmon

RFC-1147 Network Tools Directory - Groups

sourcelib

CMIP Library	map	Snmp Libraries
CMU SNMP	NETMON (III)	Snmpd (II)
HyperMIB	net_monitor	SpiderMonitor
Internet Rover	proxyd	XNETMON (II)
LANWatch	SNMP Kit	xnetperfmon

RFC-1147 Network Tools Directory - Groups

spoof

DiG
Internet Rover
mconnect
nhfsstone
nslookup
query
SPIMS

RFC-1147 Network Tools Directory - Groups

standalone

EtherMeter Sniffer SpiderMonitor

RFC-1147 Network Tools Directory - Groups

star

LAN Patrol
LANWatch
map
NETMON (III)
proxyd
Sniffer
Snmp Libraries
snmpd (II)
XNETMON (II)
xnetperfmon

RFC-1147 Network Tools Directory - Groups

status

CMIP Library
CMU SNMP
ConnectVIEW
DiG
Dual Manager
getone
Internet Rover
LanProbe
mconnect
Netlabs CMOT Agent
Netlabs SNMP Agent
netmon (I)
net_monitor
NMC
NNStat
NPRV
nslookup
osimic
osimon
OverVIEW
ping
proxyd
sma
SNMP Kit
Snmp Libraries
snmpask
snmpd (I)
snmpd (II)
snmplookup
snmpperfmon
snmppoll
snmpquery
snmpstat
snmpwatch
snmpxbar
snmpxconn
snmpxmon
snmpxperf
snmpxperfmon
TokenVIEW
Unisys NCC
WIN/MGT Station
xnetmon (I)
XNETMON (II)
xnetperfmon
xup

RFC-1147 Network Tools Directory - Groups

traffic

ENTM
etherfind
EtherMeter
EtherView
LAN Patrol
LanProbe
LANWatch
NETMON (II)
netwatch
Network Integrator
nfswatch
NMC
NNStat
osimon
OSITRACE
Sniffer
snmpxperfmon
SpiderMonitor
tcpdump
tcplogger
TRPT
Unisys NCC
WIN/MGT Station

RFC-1147 Network Tools Directory - Groups

UNIX

arp
CMIP Library
CMU SNMP
decaddrs
DiG
Dual Manager
etherfind
etherhostprobe
EtherView
getone
Internet Rover
map
mconnect
NETMON (II)
netstat
Network Integrator
net_monitor
nfswatch
nhfsstone
NMC
NNStat
nslookup
osilog
osimic
osimon
OSITRACE
ping
proxyd
query
SERAG
sma
SNMP Kit
Snmp Libraries
snmpask
snmpd (I)
snmpd (II)
snmplookup
snmpperfmon
snmppoll
snmpquery
snmproute
snmpset
snmpsrc
snmpstat
snmptrapd
snmpwatch
snmpxconn
snmpxmon
snmpxperf
snmpxperfmon
snmpxrtmetric
SPIMS
spray
tcpdump
tcplogger
traceroute
TRPT
TTCP
Unisys NCC
WIN/MGT Station
xnetmon (I)
XNETMON (II)
xnetperfmon

RFC-1147 Network Tools Directory - Groups

VMS

arp
ENTM
netstat
net_monitor
NPRV
nslookup
ping
Snmp Libraries
tcpdump
traceroute
TTCP
XNETMON (II)
xnetperfmon

RFC-1147 Network Tools Directory - Groups

X

Dual Manager	snmpxmon	WIN/MGT Station
map	snmpxperf	XNETMON (II)
snmpxbar	snmpxperfmon	xnetperfmon
snmpxconn	snmpxrtmetric	xup

Tool Descriptions

This section is a collection of brief descriptions of tools for managing TCP/IP internets. These entries are in alphabetical order, by tool name.

The entries all follow a standard format. Immediately after the NAME of a tool are its associated KEYWORDS. Keywords are terse descriptions of the purposes or attributes of a tool. A more detailed description of a tool's purpose and characteristics is given in the ABSTRACT section. The MECHANISM section describes how a tool works. In CAVEATS, warnings about tool use are given. In BUGS, known bugs or bug-report procedures are given. LIMITATIONS describes the boundaries of a tool's capabilities. HARDWARE REQUIRED and SOFTWARE REQUIRED relate the operational environment a tool needs. Finally, in AVAILABILITY, pointers to vendors, online repositories, or other sources for a tool are given.

We deal with the problem of tool-name clashes -- different tools that have the same name -- by appending parenthetical roman numerals to the names. For example, BYU, MITRE, and SNMP Research each submitted a description of a tool called "NETMON." These tools were independently developed, are functionally different, run in different environments, and are no more related than Richard Burton the 19th century explorer and Richard Burton the 20th century actor. BYU's tool "NETMON" is listed as "NETMON (I)," MITRE's as "NETMON (II)," and the tool from SNMP Research as "NETMON (III)."

The parenthetical roman numerals reveal only the order in which the catalog editor received the tool descriptions. They should not be construed to indicate any sort of preference, priority, or rights to a tool name.

RFC-1147 Network Tools Directory

arp

Keywords

routing; ethernet, IP; UNIX, VMS; free.

Abstract

Arp displays and can modify the internet-to-ethernet address translations tables used by ARP, the address resolution protocol.

Mechanism

The arp program accesses operating system memory to read the ARP data structures.

Caveats

None.

Bugs

None known.

Limitations

Only the super user can modify ARP entries.

Hardware Required

No restrictions.

Software Required

BSD UNIX or related OS, or VMS.

Availability

Available via anonymous FTP from uunet.uu.net, in directory bsd-sources/src/etc. Available with 4.xBSD UNIX and related operating systems. For VMS, available as part of TGV MultiNet IP software package, as well as Wollongong's WIN/TCP.

RFC-1147 Network Tools Directory

CMIP Library

Keywords

alarm, control, manager, status; OSI; UNIX; free, library, sourcelib.

Abstract

The CMIP Library implements the functionality of the Common Management Information Service/Protocol as in the documents ISO DP 9595-2/9596-2 of March 1988. It can act as a building block for the construction of CMIP-based agent and manager applications.

Mechanism

The CMIP library uses ISO ROS, ACSE and ASN.1 presenta tion, as implemented in ISODE, to provide its service.

Caveats

None.

Bugs

None known.

Limitations

The M-CREATE, M-DELETE and M-ACTION protocol primitives are not implemented in this version.

Hardware Required

Developed on Sun3, tested on Sun3 and VAXStation.

Software Required

The ISODE protocol suite, BSD UNIX.

Availability

The CMIP library and related management tools built upon it, known as OSIMIS (OSI Management Information Service), are publicly available from University Col lege London, England via FTP and FTAM. To obtain information regarding a copy send email to gknight@ac.ucl.cs.uk or call +44 1 380 7366.

RFC-1147 Network Tools Directory

The CMU SNMP Distribution

Keywords

manager, status; IP; NMS, SNMP; UNIX; free, sourcelib.

Abstract

The CMU SNMP Distribution includes source code for an SNMP agent, several SNMP client applications, an ASN.1 library, and supporting documentation.

The agent compiles into about 10 KB of 68000 code. The distribution includes a full agent that runs on a Kinetics FastPath2/3/4, and is built into the KIP appletalk/ethernet gateway. The machine independent portions of this agent also run on CMU's IBM PC/AT based router.

The applications are designed to be useful in the real world. Information is collected and presented in a useful format and is suitable for everyday status moni toring. Input and output are interpreted symbolically. The tools can be used without referencing the RFCs.

Mechanism

SNMP.

Caveats

None.

Bugs

None reported. Send bug reports to sw0l+snmp@andrew.cmu.edu. ("sw0l" is "ess double-you zero ell.")

Limitations

None reported.

Hardware Required

The KIP gateway agent runs on a Kinetics FastPath2/3/4. Otherwise, no restrictions.

Software Required

The code was written with efficiency and portability in mind. The applications compile and run on the following systems: IBM PC/RT running ACIS Release 3, Sun3/50 running SUNOS 3.5, and the DEC microVax running Ultrix 2.2. They are expected to run on any system with a Berkeley socket interface.

Availability

This distribution is copyrighted by CMU, but may be used and sold without permission. Consult the copyright notices for further information. The distribution is available by anonymous FTP from the host lancaster.andrew.cmu.edu (128.2.13.21) as the files pub/cmu-snmp.9.tar, and pub/kip-snmp.9.tar. The former includes the libraries and the applications, and the latter is the KIP SNMP agent.

Please direct questions, comments, and bug reports to sw0l+snmp@andrew.cmu.edu. ("sw0l" is "ess double-you zero ell.") If you pick up this package, please send a note to the above address, so that you may be notified of future enhancements/changes and additions to the set of applications (several are planned).

RFC-1147 Network Tools Directory

Computer Security Checklist

Keywords

security; DOS.

Abstract

This program consists of 858 computer security questions divided up in thirteen sections. The program presents the questions to the user and records their responses. After answering the questions in one of the thirteen sections, the user can generate a report from the questions and the user's answers. The thirteen sections are: telecommunications security, physical access security, personnel security, systems development security, security awareness and training prac tices, organizational and management security, data and program security, processing and operations security, ergonomics and error prevention, environmental secu rity, and backup and recovery security.

The questions are weighted as to their importance, and the report generator can sort the questions by weight. This way the most important issues can be tackled first.

Mechanism

The questions are displayed on the screen and the user is prompted for a single keystroke reply. When the end of one of the thirteen sections is reached, the answers are written to a disk file. The question file and the answer file are merged to create the report file.

Caveats

None.

Bugs

None known.

Limitations

None reported.

Hardware Required

No restrictions.

Software Required

DOS operating system.

Availability

A commercial product available from:

C.D., Ltd.
P.O. Box 58363
Seattle, WA 98138
(206) 243-8700

RFC-1147 Network Tools Directory

ConnectVIEW

Keywords

control, manager, routing, security, status; bridge, ethernet, ring; NMS, proprietary; DOS.

Abstract

The ConnectVIEW Network Management System consists of various software managers that control and manage Halley System's internets made of of ConnectLAN 100 ether net and ConnectLAN 200 Token Ring Brouters. The management software provides an icon-based graphical network display with real-time monitoring and reporting, along with configuration, fault, performance and security management functions for managing ConnectLAN brouters. A Planning function is also provided that allows users to draw their networks.

Mechanism

Proprietary.

Caveats

The ConnectVIEW software must be running under Microsoft Windows, preferably on a dedicated management sta tion. There is, however, no degradation of LAN throughput.

Bugs

None known.

Limitations

Currently works only with Halley System's products.]

Hardware Required

Requires a PC/AT compatible, with 640KB RAM, EGA adapter and monitor, keyboard, mouse, and ethernet adapter.

Software Required

MSDOS 3.3 or higher. Microsoft Windows/286 version 2.1.

Availability

Commercially available from:

Halley Systems, Inc.
2730 Orchard Parkway
San Jose, CA 95134

RFC-1147 Network Tools Directory

decaddrs, decaroute, decnroute, xnsroutes, bridgetab

Keywords

manager, map, routing; bridge, DECnet; NMS, SNMP; UNIX

Abstract

These commands display private MIB information from Wellfleet systems. They retrieve and format for display values of one or several MIB variables from the Wellfleet Communications private enterprise MIB, using the SNMP (RFC1098). In particular these tools are used to examine the non-IP modules (DECnet, XNS, and Bridg ing) of a Wellfleet system. Decaddrs displays the DECnet configuration of a Wellfleet system acting as a DECnet router, showing the static parameters associated with each DECnet interface. Decaroute and decnroute display the DECnet inter-area and intra-area routing tables (that is area routes and node routes). Xnsroutes displays routes known to a Wellfleet system acting as an XNS router. Bridgetab displays the bridge forwarding table with the disposition of traffic arriving from or directed to each station known to the Wellfleet bridge module. All these commands take an IP address as the argument and can specify an SNMP community for the retrieval. One SNMP query is performed for each row of the table. Note that the Wellfleet system must be operating as an IP router for the SNMP to be accessible.

Mechanism

Management information is exchanged by use of SNMP.

Caveats

None.

Bugs

None known.

Limitations

None reported.

Hardware Required

Distributed and supported for Sun 3 systems.

Software Required

Distributed and supported for SunOS 3.5 and 4.x.

Availability

Commercial product of:

Wellfleet Communications, Inc.
12 DeAngelo Drive
Bedford, MA 01730-2204
(617) 275-2400

RFC-1147 Network Tools Directory

DiG

Keywords

status; DNS; spoof; UNIX; free.

Abstract

DiG (domain information groper), is a command line tool which queries DNS servers in either an interactive or a batch mode. It was developed to be more convenient/flexible than nslookup for gathering performance data and testing DNS servers.

Mechanism

Dig is built on a slightly modified version of the bind resolver (release 4.8).

Caveats

none.

Bugs

None known.

Limitations

None reported.

Hardware Required

No restrictions.

Software Required

BSD UNIX.

Availability

DiG is available via anonymous FTP from venera.isi.edu in pub/dig.1.0.tar.Z.

RFC-1147 Network Tools Directory

Dual Manager

Keywords

alarm, control, manager, map, security, status; IP, OSI; NMS, SNMP, X; UNIX; library.

Abstract

Netlabs' Dual Manager provides management of TCP/IP networks using both SNMP and CMOT protocols. Such management can be initiated either through the X Windows user interface (both Motif and Openlook), or through OSI Network Management (CMIP) commands. The Dual Manager provides for configuration, fault, security and performance management. It provides extensive map management features, including scanned maps in the background. It provides simple mechanisms to extend the MIB and assign specific lists of objects to specific network elements, thereby providing for the management of all vendors' specific MIB extensions. It provides an optional relational DBMS for storing and retrieving MIB and alarm information. Finally, the Dual Manager is an open platform, in that it provides several Application Programming Interfaces (APIs) for users to extend the functionality of the Dual Manager.

The Dual Manager is expected to work as a TCP/IP "branch manager" under DEC's EMA, AT&T's UNMA and other OSI-conformant enterprise management architectures.

Mechanism

The Netlabs Dual Manager supports the control and monitoring of network resources by use of both CMOT and SNMP message exchanges.

Caveats

None.

Bugs

None known.

Limitations

None reported.

Hardware Required

Runs on Sun/3 and Sun/4s.

Software Required

Available on System V or SCO Open Desktop environments. Uses X-Windows for the user interface.

Availability

Commercially available from:

Netlabs Inc
11693 Chenault Street Ste 348
Los Angeles CA 90049
(213) 476-4070
lam@netlabs.com (Anne Lam)

RFC-1147 Network Tools Directory

ENTM -- Ethernet Traffic Monitor

Keywords

traffic; ethernet, IP; eavesdrop; VMS; free.

Abstract

ENTM is a screen-oriented utility that runs under VAX/VMS. It monitors local ethernet traffic and displays either a real time or cumulative, histogram showing a percent breakdown of traffic by ethernet protocol type. The information in the display can be reported based on packet count or byte count. The percent of broadcast, multicast and approximate lost pack ets is reported as well. The screen display is updated every three seconds. Additionally, a real time, sliding history window may be displayed showing ethernet traffic patterns for the last five minutes.

ENTM can also report IP traffic statistics by packet count or byte count. The IP histograms reflect information collected at the TCP and UDP port level, includ ing ICMP type/code combinations. Both the ethernet and IP histograms may be sorted by ASCII protocol/port name or by percent-value. All screen displays can be saved in a file for printing later.

Mechanism

This utility simply places the ethernet controller in promiscuous mode and monitors the local area network traffic. It preallocates 10 receive buffers and attempts to keep 22 reads pending on the ethernet dev ice.

Caveats

Placing the ethernet controller in promiscuous mode may severly slow down a VAX system. Depending on the speed of the VAX system and the amount of traffic on the local ethernet, a large amount of CPU time may be spent on the Interrupt Stack. Running this code on any production system during operational hours is discouraged.

Bugs

Due to a bug in the VAX/VMS ethernet/802 device driver, IEEE 802 format packets may not always be detected. A simple test is performed to "guess" which packets are in IEEE 802 format (DSAP equal to SSAP). Thus, some DSAP/SSAP pairs may be reported as an ethernet type, while valid ethernet types may be reported as IEEE 802 packets.

In some hardware configurations, placing an ethernet controller in promiscuous mode with automatic-restart enabled will hang the controller. Our VAX 8650 hangs running this code, while our uVAX IIs and uVAX IIIs do not.

Please report any additional bugs to the author at:

Allen Sturtevant
National Magnetic Fusion Energy Computer Center
Lawrence Livermore National Laboratory
P.O. Box 808; L-561
Livermore, CA 94550
Phone : (415) 422-8266
E-Mail: sturtevant@ccc.nmfecc.gov

Limitations

The user is required to have PHY_IO, TMPMBX and NETMBX privileges. When activated, the program first checks that the user process as enough quotas remaining (BYTLM, BIOLM, ASTLM and PAGFLQUO) to successfully run the program without entering into an involuntary wait state. Some quotas require a fairly generous setting.

The contents of IEEE 802 packets are not examined. Only the presence of IEEE 802 packets on the wire is reported.

The count of lost packets is approximated. If, after each read completes on the ethernet device, the utility detects that it has no reads pending on that device, the lost packet counter is incremented by one.

When the total number of bytes processed exceeds 7fffffff hex, all counters are automatically reset to zero.

Hardware Required

A DEC ethernet controller.

Software Required

VAX/VMS version V5.1+.

Availability

For executables only, FTP to the ANONYMOUS account (password GUEST) on CCC.NMFECC.GOV and GET the following files:

[ANONYMOUS.PROGRAMS.ENTM]ENTM.DOC (ASCII text)
[ANONYMOUS.PROGRAMS.ENTM]ENTM.EXE (binary)
[ANONYMOUS.PROGRAMS.ENTM]EN_TYPES.DAT (ASCII text)
[ANONYMOUS.PROGRAMS.ENTM]IP_TYPES.DAT (ASCII text)

RFC-1147 Network Tools Directory

etherfind

Keywords

traffic; ethernet, IP, NFS; eavesdrop; UNIX.

Abstract

Etherfind examines the packets that traverse a network interface, and outputs a text file describing the traffic. In the file, a single line of text describes a single packet: it contains values such as protocol type, length, source, and destination. Etherfind can print out all packet traffic on the ethernet, or traffic for the local host. Further packet filtering can be done on the basis of protocol: IP, ARP, RARP, ICMP, UDP, ND, TCP, and filtering can also be done based on the source, destination addresses as well as TCP and UDP port numbers.

Mechanism

In usual operations, and by default, etherfind puts the interface in promiscuous mode. In 4.3BSD UNIX and related OSs, it uses a Network Interface Tap (NIT) to obtain a copy of traffic on an ethernet interface.

Caveats

None.

Bugs

None known.

Limitations

Minimal protocol information is printed. Can only be run by the super user. The syntax is painful.

Hardware Required

Ethernet.

Software Required

SunOS.

Availability

Executable included in Sun OS "Networking Tools and Programs" software installation option.

RFC-1147 Network Tools Directory

etherhostprobe

Keywords

map, routing; ethernet, IP; ping; UNIX; free.

Abstract

Output list of hosts on an ethernet that respond to IP ARP. Produces a list in the following format:

```
08:00:20:01:96:62    128.18.4.114     apptek4
08:00:20:00:02:fe    128.18.4.115     apptek5
08:00:20:00:57:6a    128.18.4.116     apptek6
08:00:20:00:65:34    128.18.4.117     apptek7
08:00:20:06:58:6f    128.18.4.118     apptek8
08:00:20:00:03:4f    128.18.4.119     apptek9
```

The first column is the ethernet address, the second the IP address, and the third is the hostname (which is omitted if the name could not be found via gethost byaddr). A starting and ending IP address may be specified on the command line, which will limit the search.

Mechanism

Etherhostprobe sends a UDP packet to the ``echo'' port, then looks in the kernel's ARP cache for the corresponding address entry. Explicit response (or lack of same) to the UDP packet is ignored. The cache will be checked up to four times at one-quarter-second intervals. Note that this allows the program to be run by a user with no special privileges.

Caveats

Etherhostprobe will fill the kernel's ARP cache with possibly useless entries, possibly causing delays to programs foolishly attempting to accomplish real work.

Etherhostprobe causes -lots- of ARPs to be generated, possibly fooling network monitoring software (or people) into concluding that something is horribly broken.

Etherhostprobe spends up to one second looking for each possible address. Thus, exhaustively searching a class-C network will take about four minutes, and exhaustively searching a class-B network will take about 18 hours. Exhaustively searching a class-A network will take the better part of a year, so don't even think about it.

Etherhostprobe will be fooled by gateways that implement proxy ARP; every possible address on the proxy ARPed subnet will be listed with the gateway's ethernet address.

Bugs

None known.

Limitations

If a given machine is not running IP ARP at the time that it is probed, it will be considered nonexistent. In particular, if a given machine is down at the time that it is probed . . .

All hosts being probed must be on the same (possibly bridged) ethernet.

Hardware Required

No restrictions, but see below.

Software Required

Runs on SunOS 3.5, and possibly elsewhere. The major non-standard portion of code is ``tx_arp.c'', which reads the kernel's ARP cache.

Availability

Copyrighted, but freely distributed. Available via anonymous FTP from spam.itstd.sri.com (128.18.10.1). From pub directory, file EHP.1 for etherhostprobe, and files IPF.1 and IPF.2 for ipForwarding.

RFC-1147 Network Tools Directory

EtherMeter (tm), model LANB/150

Keywords

alarm, map, traffic; ethernet; NMS, proprietary; stan dalone.

Abstract

The Network Applications Technology (NAT) EtherMeter product is a dedicated ethernet traffic monitor that provides statistics on the ethernet segment to which it is attached. The EtherMeter reports three major kinds of statistics. For good packets, it reports the total number of good packets seen on the segment, the number of multicast and broadcast packets, and the total number of bytes in all packets seen. For packets with errors, it reports the number of CRC errors, short packets, oversize packets, and alignment errors. It also reports the distribution of packet by type, and the number of protocols seen on the segment. A count of transmit collisions is reported. Peak and current ethernet utilization rates are also reported, etc. Alarms can be set for utilization rate, packet rate, total error count, and delta error.

The EtherMeter reports the statistics to a Network Management Station (NMS), also available from NAT, via IP/UDP datagrams, so that the meters can be monitored through routers. The NMS displays graphical and/or textual information, and EtherMeter icons turn colors to indicate status. Alarms can be set, and if the levels are exceeded an audible alarm is generated on the NMS, and the EtherMeter icon changes from green to yellow on the network map.

Mechanism

The EtherMeter is a self-contained board that can either be plugged into a PC/AT bus for power or installed in a small stand-alone enclosure. The board can be obtained with either a 10BASE5 thick ethernet transceiver cable connector, or a 10BASE2 thin ethernet BNC connector.

Caveats

The EtherMeter is primarily a passive device whose only impact on the network will come from the monitoring packets sent to the NMS. The EtherMeter is assigned an IP address for communication with the NMS.

Bugs

None known.

Limitations

Proprietary protocol currently in use. The company has stated its intention to develop SNMP for the EtherMeter product in the first half of 1990. Currently the NMS does not keep log files. This limitation is acknowledged, and plans are underway to add ASCII log file capability to the NMS.

Hardware Required

An EtherMeter board and a PC/AT bus to plug it into, or a stand-alone enclosure with power supply (available from NAT). A Network Management Station and its software is required as well, to fully interact with the EtherMeter devices.

Software Required

The EtherMeter software is included in ROM on the dev ice. The NMS software is bundled in with the NMS hardware.

Availability

The EtherMeter device, stand-alone enclosure, and Net work Management Station, are available commercially from:

Network Application Technology, Inc.
21040 Homestead Road
Cupertino, California 95014
Phone: (408) 733-4530
Fax: (408) 733-6478

RFC-1147 Network Tools Directory

EtherView(tm)

Keywords

traffic; ethernet, IP, NFS; eavesdrop; UNIX.

Abstract

EtherView is a network monitoring tool which runs on Sun workstations and allows you to monitor your hetero- geneous internet network. It monitors all systems on the ethernet. It has three primary functions:

Load Profile: It allows users to monitor the load on the ethernet over extended periods of time. The network administrator can use it to characterize load generated by a node on the network, determine which systems and applications generate how much of the load and how that load fluctuates over long periods of time.

NFS Profile: It allows the network administrator to determine the load on NFS servers, the average response time NFS servers and the mix of NFS load on each of the servers. Users can use the data to benchmark different NFS servers, determine which servers are overloaded, deduce the number of clients that each server can support and evaluate the effectiveness of NFS accelerators.

Protocol Analyzer: Users can capture packets based on source, destination, application, protocol, bit pattern, packet size or a boolean filtering expression. It provides all standard features such as configurable buffer size, packet slicing and bit pattern based triggering criterion. It does automatic disassembly of NFS, TCP, UDP, IP, ICMP, ARP and RARP packets. Packets can be examined in any combination of summary, hex or detail format.

Mechanism

EtherView uses the Sun's NIT interface to turn the ethernet interface into promiscuous mode to capture pack ets. A high level process manages the interface and a low level process does the actual capturing and filtering. Shared memory is used to communicate between the two processes.

Bugs

None known.

Limitations

Because of limitations in Sun's NIT interface, EtherView will not capture packets originating from the system where it is run.

EtherView requires super-user privileges on the system where it is run.

Hardware Required

EtherView runs on all models of Sun-3, Sun-4 and Sun 386i.

Software Required

Sun-3 - SunOS 4.0.3. (SunOS 4.0 with NIT fixes).
Sun-4 - SunOS 4.0.
Sun-386i - SunOS 4.0.

Runs under SunView.

Will run under X Windows in future.

Availability

EtherView is copyrighted, commercial product of:

Matrix Computer Systems, Inc.
7 1/2 Harris Road
Nashua, NH 03062

Tel: (603) 888-7790
email: ...uunet!matrix!eview

RFC-1147 Network Tools Directory

getone, getmany, getroute, getarp, getaddr, getif, getid.

Keywords

manager, routing, status; IP; NMS, SNMP; UNIX.

Abstract

These commands retrieve and format for display values of one or several MIB variables (RFC1066) using the SNMP (RFC1098). Getone and getmany retrieve arbitrary MIB variables; getroute, getarp, getaddr, and getif retrieve and display tabular information (routing tables, ARP table, interface configuration, etc.), and getid retrieves and displays system name, identification and boot time.

Getone <target> <mibvariable> retrieves and displays the value of the designated MIB variable from the specified target system. The SNMP community name to be used for the retrieval can also be specified. Getmany works similarly for groups of MIB variables rather than individual values. The name of each variable, its value and its data type is displayed. Getroute returns information from the ipRoutingTable MIB structure, displaying the retrieved information in an accessible format. Getarp behaves similarly for the address translation table; getaddr for the ipAddressTable; and getif displays information from the interfaces table, supplemented with information from the ipAddressTable. Getid displays the system name, identification, ipForwarding state, and the boot time and date. All take a system name or IP address as an argument and can specify an SNMP community for the retrieval. One SNMP query is performed for each row of the table.

Mechanism

Queries SNMP agent(s).

Caveats

None.

Bugs

None known.

Limitations

None reported.

Hardware Required

Distributed and supported for Sun 3 systems.

Software Required

Distributed and supported for SunOS 3.5 and 4.x.

Availability

Commercial product of:

Wellfleet Communications, Inc.
12 DeAngelo Drive
Bedford, MA 01730-2204
(617) 275-2400

RFC-1147 Network Tools Directory

hammer & anvil

Keywords

benchmark, generator; IP; DOS; free.

Abstract

Hammer and anvil are the benchmarking programs for IP routers. Using these tools, gateways have been tested for per-packet delay, router-generated traffic overhead, maximum sustained throughput, etc.

Mechanism

Tests are performed on a gateway in an isolated testbed. Hammer generates packets at controlled rates. It can set the length and interpacket interval of a packet stream. Anvil counts packet arrivals.

Caveats

Hammer should not be run on a live network.

Bugs

None reported.

Limitations

Early versions of hammer could not produce inter-packet intervals shorter than 55 usec.

Hardware Required

Hammer runs on a PC/AT or compatible, and anvil requires a PC or clone. Both use a Micom Interlan NI5210 for LAN interface.

Software Required

MS-DOS.

Availability

Hammer and anvil are copyrighted, though free. Copies are available from pub/eutil on husc6.harvard.edu.

RFC-1147 Network Tools Directory

hopcheck

Keywords

routing; IP; ping; DOS; free.

Abstract

Hopcheck is a tool that lists the gateways traversed by packets sent from the hopcheck-resident PC to a desti nation. Hopcheck uses the same mechanism as traceroute but is for use on IBM PC compatibles that have ethernet connections. Hopcheck is part of a larger TCP/IP package that is known as ka9q that is for use with packet radio. Ka9q can coexist on a PC with other TCP/IP packages such as FTP Inc's PC/TCP, but must be used independently of other packages. Ka9q was written by Phil Karn. Hopcheck was added by Katie Stevens, dkstevens@ucdavis.edu. Unlike traceroute, which requires a UNIX kernel mod, hopcheck will run on the standard, unmodified ka9q release.

Mechanism

See the description in traceroute.

Caveats

See the description in traceroute.

Bugs

None known.

Limitations

Host table required. Does not work with domain name server or with IP address as the argument. This is mainly an inconvenience.

Hardware Required

IBM PC compatible with ethernet network interface card, though does not work with 3Com 505 board.

Software Required

DOS.

Availability

Free. On deposit at the National Center for Atmospher ic Research. For access from UNIX, available via anonymous FTP from windom.ucar.edu, in directory "etc," as hopcheck.tar.Z. For access directly from a PC, fetch nethop.exe and readme.hop; nethop.exe is execut able. Also available via anonymous FTP at ucdavis.edu, in the nethopexe or nethopsrc suite of files in direc tory "dist."

RFC-1147 Network Tools Directory

HyperMIB

Keywords

reference; Macintosh; free, sourcelib.

Abstract

HyperMIB is a hypertext presentation of the MIB (RFC1066). The tree structure of the MIB is presented graphically, and the user traverses the tree by selecting branches of the tree. When the MIB variables are displayed, selecting them causes a text window to appear and show the definition of that variable (using the actual text of the MIB document).

Mechanism

The Apple Macintosh HyperCard utility is used. The actual text of the MIB document is read into scrollable text windows, and a string search is done on the variable selected. A person familiar with HyperCard programming could modify the program to suit their needs (such as to add the definitions for their company's private space).

Caveats

None.

Bugs

None known.

Limitations

This program only gives the definition of the MIB variables. It cannot poll a node to find the value of the variables.

Hardware Required

Apple Macintosh computer with at least 1MByte of RAM.

Software Required

Apple Macintosh operating system and HyperCard.

Availability

This software may be copied and given away without charge. The files are available by anonymous FTP on CCC.NMFECC.GOV. The files are:

[Anonymous.programs.HyperMIB]Hyper_MIB.help (ASCII text)
[Anonymous.programs.HyperMIB]Hyper.MIB (binary)
[Anonymous.programs.HyperMIB]MIB.tree (binary)

The software is also available for a nominal fee from:

National Energy Software Center
Argonne National Laboratory
9700 South Cass Avenue
Argonne, Illinois 60439
(312) 972-7250

RFC-1147 Network Tools Directory

Internet Rover

Keywords

status; IP, SMTP; curses, ping, spoof; UNIX; free, sourcelib.

Abstract

Internet Rover is a prototype network monitor that uses multiple protocol "modules" to test network functional ity. This package consists of two primary pieces of code: the data collector and the problem display.

There is one data collector that performs a series of network tests, and maintains a list of problems with the network. There can be many display processes all displaying the current list of problems which is useful in a multi-operator NOC.

The display task uses curses, allowing many terminal types to display the problem file either locally or from a remote site. Full source is provided. The data collector is easily configured and extensible. Contributions such as additional protocol modules, and shell script extensions are welcome.

Mechanism

A configuration file contains a list of nodes, addresses, NodeUp? protocol test (ping in most cases), and a list of further tests to be performed if the node is in fact up. Modules are included to test TELNET, FTP, and SMTP. If the configuration contains a test that isn't recognized, a generic test is assumed, and a filename is checked for existence. This way users can create scripts that create a file if there is a problem, and the data collector simply checks the existence of that file to determine if there is problem.

Caveats

None.

Bugs

None known.

Limitations

This tools does not yet have the capability to perform actions based on the result of the test. Rather, it is intended for a multi-operator environment, and simply displays a list of what is wrong with the net.

Hardware Required

This software is known to run on Suns and IBM RTs.

Software Required

Curses, 4.xBSD UNIX socket programming libraries, BSD ping.

Availability

Full source available via anonymous FTP from merit.edu (35.1.1.42) in the ~ftp/pub/inetrover directory. Source and executables are public domain and can be freely distributed for non-commercial use. This package is unsupported, but bug reports and fixes may be sent to: wbn@merit.edu.

RFC-1147 Network Tools Directory

LAN Patrol

Keywords

security, traffic; ethernet, star; eavesdrop; DOS.

Abstract

LAN Patrol is a full-featured network analyzer that provides essential information for effective fault and performance management. It allows network managers to easily monitor user activity, find traffic overloads, plan for growth, test cable, uncover intruders, balance network services, and so on. LAN Patrol uses state of the art data collection techniques to monitor all activity on a network, giving an accurate picture of how it is performing.

LAN Patrol's reports can be saved as ASCII files to disk, and imported into spreadsheet or database programs for further analysis.

Mechanism

The LAN Patrol interface driver programs a standard interface card to capture all traffic on a network seg ment. The driver operates from the background of a standard PC, maintaining statistics for each station on the network. The information can be viewed on the PC's screen, or as a user-defined report output either to file or printer.

Caveats

None. Normal operation is completely passive, making LAN Patrol transparent to the network.

Bugs

None known.

Limitations

LAN Patrol can monitor up to 10,000 packets/sec on an AT class PC, and is limited to monitoring a maximum of 1024 stations for intervals of up to 30 days.

Because LAN Patrol operates at the physical level, it will only see traffic for the segment on which it is installed; it cannot see traffic across bridges.

Hardware Required

Computer: IBM PC/XT/AT, PS/2 Model 30, or compatible. Requires 512K memory and a hard drive or double-sided disk drive.

Display: Color or monochrome text. Color display allows color-coding of traffic information.

Ethernet, StarLAN, LattisNet, or StarLAN 10 network interface card.

Software Required

PC DOS, MS-DOS version 3.1 or greater.

Availability

LAN Patrol many be purchased through network dealers, or directly from:

Legend Software, Inc.
Phone: (201) 227-8771
FAX: (201) 906-1151

RFC-1147 Network Tools Directory

LanProbe -- the HP 4990S LanProbe Distributed Analysis System.

Keywords

alarm, manager, map, status, traffic; ethernet; eaves drop, NMS; proprietary.

Abstract

The LanProbe distributed monitoring system performs remote and local monitoring of ethernet LANs in a protocol and vendor independent manner.

LanProbe discovers each active node on a segment and displays it on a map with its adapter card vendor name, ethernet address, and IP address. Additional information about the nodes, such as equipment type and physical location can be entered in to the data base by the user.

When the NodeLocator option is used, data on the actual location of nodes is automatically entered and the map becomes an accurate representation of the physical layout of the segment. Thereafter when a new node is installed and becomes active, or when a node is moved or becomes inactive, the change is detected and shown on the map in real time. The system also provides the network manager with precise cable fault information displayed on the map.

Traffic statistics are gathered and displayed and can be exported in (comma delimited) CSV format for further analysis. Alerts can be set on user defined thres holds.

Trace provides a remote protocol analyzer capability with decodes for common protocols.

Significant events (like power failure, cable breaks, new node on network, broadcast IP source address seen, etc.) are tracked in a log that is uploaded to ProbeView periodically.

ProbeView generates reports that can be manipulated by MSDOS based word processors, spreadsheets, and DBMS.

Mechanism

The system consists of one or more LanProbe segment monitors and ProbeView software running under Microsoft Windows. The LanProbe segment monitor attaches to the end of an ethernet segment and monitors all traffic. Attachment can be direct to a thin or thick coax cable, or via an external transceiver to fiber optic or twisted pair cabling. Network data relating to the segment is transferred to a workstation running ProbeView via RS-232, ethernet, or a modem connection.

ProbeView software, which runs on a PC/AT class workstation, presents network information in graphical displays.

The HP4992A NodeLocator option attaches to the opposite end of the cable from the HP4991A LanProbe segment mon itor. It automatically locates the position of nodes on the ethernet networks using coaxial cabling schemes.

Caveats

None.

Bugs

None known.

Limitations

None reported.

Hardware Required

HP 4991A LanProbe segment monitor
HP 4992A NodeLocator (for optional capabilities)
80386 based PC capable of running MS-Windows

Software Required

HP 4990A ProbeView
MSDOS 3.0 or higher and Microsoft Windows/286 2.1.

Availability

A commercial product available from:

Hewlett-Packard Company
P.O. Box 10301,
Palo Alto, CA 94303-0890

RFC-1147 Network Tools Directory

LANWatch

Keywords

alarm, analyzer, traffic; CHAOS, DECnet, DNS, ethernet, IP, OSI, ring, SMTP, star; eavesdrop; DOS; library, sourcelib.

Abstract

LANWatch 2.0 is an inexpensive, powerful and flexible network analyzer that runs under DOS on personal computers and requires no hardware modifications to either the host or the network. LANWatch is an invaluable tool for installing, troubleshooting, and monitoring local area networks, and for developing and debugging new protocols. Network managers using LANWatch can inspect network traffic patterns and packet errors to isolate performance problems and bottlenecks. Protocol developers can use LANWatch to inspect and verify proper protocol handling. Since LANWatch is a software-only package which installs easily in existing PCs, network technicians and field service engineers can carry LANWatch in their briefcase for convenient network analysis at remote sites.

LANWatch has two operating modes: Display and Examine. In Display Mode, LANWatch traces network traffic by displaying captured packets in real time. Examine Mode allows you to scroll back through stored packets to inspect them in detail. To select a subset of packets for display, storage or retrieval, there is an extensive set of built-in filters. Using filters, LANWatch collects only packets of interest, saving the user from having to sort through all network traffic to isolate specific packets. The built-in filters include alarm, trigger, capture, load, save and search. They can be controlled separately to match on source or destination address, protocol, or packet contents at the hardware and transport layers. LANWatch also includes sufficient source code so users can modify the existing filters and parsers or add new ones.

The LANWatch distribution includes executables and source for several post-processors: a TCP protocol analyzer, a node-by-node traffic analyzer and a dump file listing tool.

Mechanism

Uses many common PC network interfaces by placing them in promiscuous mode and capturing traffic.

Caveats

Most PC network interfaces will not capture 100% of the traffic on a fully-loaded network (primarily missing back-to-back packets).

Bugs

None known.

Limitations

LANWatch can't analyze what it doesn't see (see Caveats).

Hardware Required

LANWatch requires a PC or PS/2 with a supported network interface card.

Software Required

LANWatch runs in DOS. Modification of the supplied source code or creation of additional filters and parsers requires Microsoft C 5.1

Availability

LANWatch is commercially available from:

FTP Software, Incorporated
26 Princess Street
Wakefield, MA, 01880
(617 246-0900).

RFC-1147 Network Tools Directory

map -- Interactive Network Map

Keywords

manager, map; CHAOS, ethernet, IP, ring, star; NMS, ping, SNMP, X; UNIX; free, sourcelib.

Abstract

Map draws a map of network connectivity and allows interactive examination of information about various components including whether hosts can be reached over the network.

The program is supplied with complete source and is written in a modular fashion to make addition of different protocols stacks, displays, or hardcopy devices relatively easy. This is one of the reasons why the initial version supports at least two of each. Contributions of additional drivers in any of these areas will be welcome as well as porting to additional platforms.

Mechanism

Net components are pinged by use of ICMP echo and, optionally, CHAOS status requests and SNMP "gets." The program initializes itself from static data stored in the file system and therefore does not need to access the network in order to get running (unless the static files are network mounted).

Caveats

As of publication, the tool is in beta release.

Bugs

Several minor nits, documented in distribution files. Bug discoveries should be reported by email to Bug Map@LCS.MIT.Edu.

Limitations

See distribution file for an indepth discussion of system capabilities and potential.

Hardware Required

An X display is needed for interactive display of the map, non-graphical interaction is available in nondisplay mode. For hardcopy output a PostScript or Tek tronix 4692 printer is required.

Software Required

BSD UNIX or related OS. IP/ICMP is required; CHAOS/STATUS and SNMP can be used but are optional. X-Windows is required for interactive display of the map.

Availability

As of publication, map is in beta release. To be added to the email forum that discusses the software, or to obtain individual files or instructions on getting the full current release, send a request to:

MAP-Request@LCS.MIT.Edu.

RFC-1147 Network Tools Directory

mconnect

Keywords

status; SMTP; spoof; UNIX.

Abstract

Mconnect allows an interactive session with a remote mailer. Mail delivery problems can be diagnosed by connecting to the remote mailer and issuing SMTP commands directly.

Mechanism

Opens a TCP connection to remote SMTP on port 25. Provides local line buffering and editing, which is the distinction between mconnect and a TELNET to port 25.

Caveats

None.

Bugs

None known.

Limitations

Mconnect is not a large improvement over using a TELNET connection to port 25.

Hardware Required

No restrictions.

Software Required

BSD UNIX or related OS.

Availability

Available with 4.xBSD UNIX and related operating sys tems.

RFC-1147 Network Tools Directory

Netlabs CMOT Agent

Keywords

manager, status; IP, OSI; NMS.

Abstract

Netlabs' CMOT code debuted in Interop 89. The CMOT code comes with an Extensible MIB, which allows users to add new MIB variables. The code currently supports all the MIB variables in RFC 1095 via the data types in RFC 1065, as well as the emerging MIB-II, which is currently in experimental stage. The CMOT has been benchmarked at 100 Management Operations per Second (MOPS) for a 1-MIPS machine.

Mechanism

The Netlabs CMOT agent supports the control and monitoring of network resources by use of CMOT message exchanges.

Caveats

None.

Bugs

None known.

Limitations

None reported.

Hardware Required

Portable to most hardware.

Software Required

Portable to most operating systems.

Availability

Commercially available from:

Netlabs Inc
11693 Chenault Street Ste 348
Los Angeles CA 90049
(213) 476-4070
lam@netlabs.com (Anne Lam)

RFC-1147 Network Tools Directory

Netlabs SNMP Agent.

Keywords

manager, status; IP; NMS, SNMP.

Abstract

Netlabs' SNMP code debuted in Interop 89, where it showed interoperation of the code with several implementations on the show floor. The SNMP code comes with an Extensible MIB, which allows users to add new MIB variables. The code currently supports all the MIB variables in RFC 1066 via the data types in RFC 1065, as well as the emerging MIB-II, which is currently in experimental stage. The SNMP has been benchmarked at 200 Management Operations per Second (MOPS) for a 1 MIPS machine.

Mechanism

The Netlabs SNMP agent supports the control and monitoring of network resources by use of SNMP message exchanges.

Caveats

None.

Bugs

None known.

Limitations

None reported.

Hardware Required

Portable to most hardware.

Software Required

Portable to most operating systems.

Availability

Commercially available from:

Netlabs Inc
11693 Chenault Street Ste 348
Los Angeles CA 90049
(213) 476-4070
lam@netlabs.com (Anne Lam)

RFC-1147 Network Tools Directory

netmon (I)

Keywords

status; DNS, IP; ping; DOS; free.

Abstract

Netmon is a DOS-based program that pings hosts on a monitored list at user-specified intervals. In addition, a user may optionally ping hosts not on the list.

Netmon also performs domain lookups. Furthermore, a user may build and send a domain query to any desired DNS server.

Mechanism

The tool works by using the echo service feature of ICMP. It reports if it receives an incorrect response or no response.

Caveats

Depending on the frequency of pinging and the number of hosts pinged, netmon could create a high volume of traffic.

Bugs

None known.

Limitations

None reported.

Hardware Required

A PC, and a Western Digital WD8003 interface card (or any other card for which there is a packet driver for FTP Software Inc.'s PC/TCP kernel). Both monochrome and color displays are supported, though color is recommended.

Software Required

DOS operating system, and the PC/TCP Kernel by FTP Software, Inc.

Availability

The BYU modified version is available for anonymous FTP from Dcsprod.byu.edu, in directory "programs." It can be freely distributed for non-commercial use.

RFC-1147 Network Tools Directory

NETMON (II)
and iptrace

Keywords

traffic; IP; eavesdrop; UNIX; free.

Abstract

NETMON is a facility to enable communication of networking events from the BSD UNIX operating system to a user-level network monitoring or management program. Iptrace is a program interfacing to NETMON which logs TCP-IP traffic for performance measurement and gateway monitoring. It is easy to build other NETMON-based tools using iptrace as a model.

NETMON resides in the 4.3BSD UNIX kernel. It is independent of hardware-specific code in UNIX. It is transparent to protocol and network type, having no internal assumptions about the network protocols being recorded. It is installed in BSD-like kernels by adding a standard function call (probe) to a few points in the input and output routines of the protocols to be logged.

NETMON is analogous to Sun Microsystems' NIT, but the interface tap function is extended by recording more context information. Aside from the timestamp, the choice of information recorded is up to the installer of the probes. The NETMON probes added to the BSD IP code supplied with the distribution include as context: input and output queue lengths, identification of the network interface, and event codes labeling packet dis cards. (The NETMON distribution is geared towards measuring the performance of BSD networking protocols in an IP gateway).

NETMON is designed so that it can reside within the monitored system with minimal interference to the network processing. The estimated and measured overhead is around five percent of packet processing.

The user-level tool "iptrace" is provided with NETMON. This program logs IP traffic, either at IP-level only, or as it passes through the network interface drivers as well. As a separate function, iptrace produces a host traffic matrix output. Its third type of output is abbreviated sampling, in which only a pre-set number of packets from each new host pair is logged. The three output types are configured dynamically, in any combination.

OSITRACE, another logging tool with a NETMON interface, is available separately (and documented in a separate entry in this catalog).

Mechanism

Access to the information logged by NETMON is through a UNIX special file, /dev/netmon. User reads are blocked until the buffer reaches a configurable level of full ness.

Several other parameters of NETMON can be tuned at compile time. A diagnostic program, netmonstat, is included in the distribution.

Caveats

None.

Bugs

Bug reports and questions should be addressed to:

ie-tools@gateway.mitre.org

Requests to join this mailing list:

ie-tools-request@gateway.mitre.org

Questions and suggestions can also be directed to:

Allison Mankin (703)883-7907
mankin@gateway.mitre.org

Limitations

A NETMON interface for tcpdump and other UNIX protocol analyzers is not included, but it is simple to write. NETMON probes for a promiscuous ethernet interface are similarly not included.

Hardware Required

No restrictions.

Software Required

BSD UNIX-like network protocols or the ability to install the BSD publicly available network protocols in the system to be monitored.

Availability

The NETMON distribution is available by anonymous FTP in pub/netmon.tar or pub/netmon.tar.Z from aelred 3.ie.org. A short user's and installation guide, NETMON.doc, is available in the same location. The NETMON distribution is provided "as is" and requires retention of a copyright text in code derived from it. It is copyrighted by the MITRE-Washington Networking Center.

RFC-1147 Network Tools Directory

NETMON (III)
SNMP-based network management tool
from SNMP Research.

Keywords

alarm, control, manager, map, routing; DECnet, ether net, IP, OSI, ring, star; NMS, SNMP; DOS; sourcelib.

Abstract

The NETMON application implements a network management station based on a low-cost DOS-based platform. It can be successfully used with many types of networks, including both wide area networks and those based on various LAN media. NETMON has been used with multiprotocol devices including those which support TCP/IP, DECnet, and OSI protocols. The fault management tool displays the map of the network configuration with current node and link state indicated in one of several colors. Alarms may be enabled to alert the operator of events occurring in the network. Events are logged to disk. The NETMON application comes complete with source code including a powerful set of portable libraries for generating and parsing SNMP messages. Output data from NETMON may be transferred via flat files for additional report generation by a variety of statistical packages.

Mechanism

The NETMON application is based on the Simple Network Management Protocol (SNMP). Polling is performed via the powerful SNMP get-next operator and the SNMP get operator. Trap directed polling is used to regulate the focus and intensity of the polling.

Caveats

None.

Bugs

None known.

Limitations

The monitored and managed nodes must implement the SNMP over UDP per RFC 1098 or must be reachable via a proxy agent.

Hardware Required

The minimum system is a IBM Personal Computer (4.77 MHz) with DOS 3.0 or later, an Enhanced Graphics Adapter, Enhanced Graphics Monitor, a single 360 Kbyte floppy drive, and an ethernet adapter. However, most users will find a hard disk to be helpful for storing network history and will be less impatient with a fas ter CPU.

Software Required

DOS 3.0 or later and TCP/IP software from one of several sources.

Availability

This is a commercial product available under license from:

SNMP Research

P.O. Box 8593
Knoxville, TN 37996-4800
(615) 573-1434 (Voice)
(615) 573-9197 (FAX)
Attn: Dr. Jeff Case

RFC-1147 Network Tools Directory

netstat

Keywords

routing; IP; UNIX, VMS; free.

Abstract

Netstat is a program that accesses network related data structures within the kernel, then provides an ASCII format at the terminal. Netstat can provide reports on the routing table, TCP connections, TCP and UDP "listens", and protocol memory management.

Mechanism

Netstat accesses operating system memory to read the kernel routing tables.

Caveats

Kernel data structures can change while netstat is running.

Bugs

None known.

Limitations

None reported.

Hardware Required

No restrictions.

Software Required

BSD UNIX or related OS, or VMS.

Availability

Available via anonymous FTP from uunet.uu.net, in directory bsd-sources/src/ucb. Available with 4.xBSD UNIX and related operating systems. For VMS, available as part of TGV MultiNet IP software package, as well as Wollongong's WIN/TCP.

RFC-1147 Network Tools Directory

netwatch

Keywords

traffic; ethernet, IP, ring; eavesdrop; DOS; free.

Abstract

PC/netwatch listens to an attached local broadcast network and displays one line of information for every packet that goes by. This information consists of the "to" and "from" local network addresses, the packet length, the value of the protocol type field, and 8 selected contiguous bytes of the packet contents. While netwatch is running it will respond to commands to display collected information, change its operating mode, or to filter for specific types of packets.

Mechanism

Puts controller in promiscuous mode.

Caveats

None.

Bugs

None known.

Limitations

The monitor can handle a burst rate of about 200 packets per second. Packets arriving faster than that are missed (but counted in the statistics of the network driver). The display rate is about 25 packets per second and there is a buffer that can hold 512 undisplayed packets. The monitor discards overflow packets.

Hardware Required

IBM PC compatible with CGA and network interface (3com 3C501, Interlan NI5010, or proNet p1300).

Software Required

DOS 2.0 or higher, MicroSoft C (to generate custom executables)

Availability

Available as a utility program in the pcip distribution from host husc6.harvard.edu, in directory pub/pcip. Available in a standalone package via anonymous FTP from windom.ucar.edu, in file pc/network/netwatch.arc; a binary "dearc" program is also available from windom.ucar.edu.

RFC-1147 Network Tools Directory

Network Integrator I

Keywords

map, traffic; ethernet; UNIX.

Abstract

This tool monitors traffic on network segments. All information is dumped to either a log file or, for real-time viewing, to a command tool window. Data is time-stamped according to date and time. Logging can continue for up to 24 hours.

The tool is flexible in data collection and presenta tion. Traffic filters can be specified according to header values of numerous protocols, including those used by Apple, DEC, Sun, HP, and Apollo. Bandwidth utilization can be monitored, as well as actual load and peak throughput. Additionally, the Network Integrator can analyze a network's topology, and record the location of all operational nodes on a network.

Data can be displayed in six separate formats of bar graphs. In addition, there are several routines for producing statistical summaries of the data collected.

Mechanism

The tools work through RPC and XDR calls.

Caveats

Although the tool adds only little traffic to a network, generation of statistics from captured files requires a significant portion of a workstation's CPU.

Bugs

None known.

Limitations

Must be root to run monitor. There does not seem to be a limit to the number of nodes, since it monitors by segments. The only major limitation is the amount of disk space that a user can commit to the log files. The size of the log files, however, can be controlled through the tool's parameters.

Hardware Required

Sun3 or Sun4.

Software Required

4.0BSD UNIX or greater, or related OS.

Availability

Copyrighted, commercially available from

Network Integrators,
(408) 927-0412.

RFC-1147 Network Tools Directory

net_monitor

Keywords

routing, status; DECnet, IP; curses, ping; UNIX, VMS; free, sourcelib.

Abstract

Net_monitor uses ICMP echo (and DECnet reachability information on VAX/VMS) to monitor a network. The monitoring is very simplistic, but has proved useful. It periodically tests whether hosts are reachable and reports the results in a full-screen display. It groups hosts together in common sets. If all hosts in a set become unreachable, it makes a lot of racket with bells, since it assumes that this means that some common piece of hardware that supports that set has failed. The periodicity of the tests, hosts to test, and groupings of hosts are controlled with a single configuration file.

The idea for this program came from the PC/IP monitor facility, but is an entirely different program with different functionality.

Mechanism

Reachability is tested using ICMP echo facilities for TCP/IP hosts (and DECnet reachability information on VAX/VMS). A DECnet node is considered reachable if it appears in the list of hosts in a "show network" command issued on a routing node.

Caveats

This facility has been found to be most useful when run in a window on a workstation rather than on a terminal connected to a host. It could be useful if ported to a PC (looks easy using FTP Software's programming libraries), but this has not been done. Curses is very slow and cpu intensive on VMS, but the tool has been run in a window on a VAXstation 2000. Just don't try to run it on a terminal connected to a 11/750.

Bugs

None known.

Limitations

This tool is not meant to be a replacement for a more comprehensive network management facility such as is provided with SNMP.

Hardware Required

A host with a network connection.

Software Required

Curses, 4.xBSD UNIX socket programming libraries (limited set) and some flavor of TCP/IP that supports ICMP echo request (ping). It has been run on VAX/VMS running WIN/TCP and several flavors of 4BSD UNIX (including SunOS 3.2, 4.0, and 4.3BSD). It could be ported to any platform that provides a BSD-style programming library with an ICMP echo request facility and curses.

Availability

Requests should be sent to the author:

Dale Smith
Asst Dir of Network Services

University of Oregon
Computing Center
Eugene, OR 97403-1211

Internet: dsmith@oregon.uoregon.edu.
BITNET: dsmith@oregon.bitnet
UUCP: ...hp-pcd!uoregon!dsmith
Voice: (503)686-4394

With the source code, a makefile is provided for most any UNIX box and a VMS makefile compatible with the make distributed with PMDF. A VMS DCL command file is also provided, for use by those VMS sites without "make."

The author will attempt to fix bugs, but no support is promised. The tool is copyrighted, but free (for now).

RFC-1147 Network Tools Directory

nfswatch

Keywords

traffic; ethernet, IP, NFS; curses, eavesdrop; UNIX; free.

Abstract

Nfswatch monitors all incoming ethernet traffic to an NFS file server and divides it into several categories. The number and percentage of packets received in each category is displayed on the screen in a continuously updated display.

All exported file systems are monitored by default. Other files may optionally be monitored. Options also allow monitoring of traffic destined for a remote host instead of the local host, or monitoring traffic sent by a single host. Items such as the sample interval length can be adjusted either on the command line or interactively. Facilities for taking screen "snapshots," saving all data to a log file, and summarizing the log file are included. Nfslogsum, a program that summarizes the log file, is included in the distribution.

Mechanism

Nfswatch uses the Network Interface Tap in promiscuous mode to monitor the ethernet. It filters out NFS packets destined for the local (or remote) host, and then decodes the file handles in order to determine which file or file system a request pertains to.

Caveats

Because the NFS file handle is a non-standard (server private) piece of data, the file system monitoring part of the program will break whenever the format of a file handle is not what it expects to see. This is easily fixed in the code, however. The code presently under stands SunOS 4.0 file handles.

Bugs

None known.

Limitations

Up to 256 exported file systems and 256 individual files can be monitored, but only (2 * (DisplayLines 16)) will be displayed on the screen (all data will be written to the log file).

Only NFS requests made by client machines are counted; the NFS traffic generated by the server in response to these requests is not counted.

Hardware Required

Has been tested on Sun-3 and Sun-4 systems. No hardware dependencies, but see below.

Software Required

SunOS 4.0 or higher. The STREAMS NIT device is used. Fairly easy code modifications should be able to make it run under older SunOS releases, or other versions of BSD UNIX with a NIT-like device.

Availability

Copyrighted, but freely distributable. Available via anonymous FTP from hosts icarus.riacs.edu and spam.itstd.sri.com in pub/nfswatch.tar.Z. There should also be a copy on the 1989 Sun User's Group tape.

RFC-1147 Network Tools Directory

nhfsstone

Keywords

benchmark, generator; NFS; spoof; UNIX; free.

Abstract

Nhfsstone (pronounced n-f-s-stone, the "h" is silent) is an NFS benchmarking program. It is used on an NFS client to generate an artificial load with a particular mix of NFS operations. It reports the average response time of the server in milliseconds per call and the load in calls per second. The nhfsstone distribution includes a script, "nhfsnums" that converts test results into plot(5) format so that they can be graphed using graph(1) and other tools.

Mechanism

Nhfsstone is an NFS traffic generator. It adjusts its calling patterns based on the client's kernel NFS statistics and the elapsed time. Load can be generated over a given time or number of NFS calls.

Caveats

Nhfsstone will compete for system resources with other applications.

Bugs

None known.

Limitations

None reported.

Hardware Required

No restrictions.

Software Required

4.xBSD-based UNIX

Availability

Available via anonymous FTP from bugs.cs.wisc.edu. Alternatively, Legato Systems will provide the program free of charge, if certain conditions are met. Send name and both email and U.S. mail addresses to:

Legato Systems, Inc.
Nhfsstone
260 Sheridan Avenue
Palo Alto, California 94306

A mailing list is maintained for regular information and bug fixes: nhfsstone@legato.com or uunet!legato.com!nhfsstone. To join the list: nhfsstone-request@legato.com or uunet!legato.com!nhfsstone-request.

RFC-1147 Network Tools Directory

NMC -- the Hughes LAN Systems 9100
Network Management Center

Keywords

control, manager, routing, status, traffic; bridge, DECnet, ethernet, IP; NMS, SNMP; UNIX.

Abstract

The 9100 Network Management Center provides the capability to manage and control standards-based networking products from Hughes LAN Systems' and other vendors. This management extends to all network products that are equipped with the industry standard SNMP (Simple Network Management Protocol). A comprehensive relational database manages the data and ensures easy access and control of resources throughout the network.

9100 NMC software provides the following functions:

Database Management

Stores and retrieves the information required to administer and configure the network. It can be used to:

Store and recall configuration data for all

devices.

Provide availability history for devices.

Provides full-function SQL interface.

Assign new internet addresses.

Provide administrative information such as physical location of devices, person responsible, maintenance history, asset data, hardware/software versions, etc.

Configuration Management

A comprehensive configuration model that enables you to:

Retrieve configuration information from SNMP devices.

Configure HLS devices using SNMP.

Configures attributes relating to TCP/IP, DECnet and other protocols in HLS devices

using SNMP.

Poll devices to compare their current attribute values with those in the database and produce reports of the discrepancies.

Collect data about the state of the network.

Performance Management

Displays local network traffic graphically, by packet size, protocol, network utilization, sources and destinations of packets, etc.

Fault Management

Provides availability monitoring and indicates potential problems.

Scheduled availability monitoring of devices.

SNMP traps (alarms) are recorded in an alarm log.

New alarms are indicated by a flashing icon and optional audio alert.

Possible causes and suggested actions for the alarms are listed.

Cumulative reports can be produced.

Utilities Function

Allows you to view and/or stop existing NMC processes, and to define schedules for invoking

NMC applications and database maintenance utilities.

Mechanism

SNMP.

Caveats

None reported.

Bugs

None known.

Limitations

Maximum number of nodes that can be monitored is 18,000. This can include Hosts, Terminal Servers, PCs, and Bridges.

Hardware Required

The host for the NMC software is a Sun 3 desktop workstation. Recommended minimum hardware is the Sun 3/80 Color with a 1/4" SCSI tape drive.

Software Required

The NMC, which is provided on 1/4" tape format, runs on the Sun 4.0 Operating System.

Availability

A commercial product of:

Hughes LAN Systems Inc.
1225 Charleston Road
Mountain View, CA 94043
Phone: (415) 966-7300
Fax: (415) 960-3738
RCA Telex: 276572

RFC-1147 Network Tools Directory

NNStat

Keywords

manager, status, traffic; ethernet, IP; eavesdrop, NMS; UNIX; free.

Abstract

NNStat is a collection of programs that provides an internet statistic collecting capability. The NNStat strategy for statistic collection is to collect traffic statistics via a promiscuous ethernet tap on the local networks, versus instrumenting the gateways. If all traffic entering or leaving a network or set of networks traverses a local ethernet, then by stationing a statistic gathering agent on each local network a profile of network traffic can be gathered. Statistical data is retrieved from the local agents by a global manager. A program called "statspy" performs the data gathering function. Essentially, statspy reads all packets on an ethernet interface and records all information of interest. Information of interest is gathered by examining each packet and determining if the source or destination IP address is one that is being monitored, typically a gateway address. If so then the contents of the packet are examined to see if they match further criteria. A program called "collect" performs global data collection. It periodically polls various statspy processes in the domain of interest to retrieve locally logged statistical data. The NNSTAT distribution comes with several sample awk programs which process the logged output of the collect program.

Mechanism

Local agents (statspy processes) collect raw traffic data via a promiscuous ethernet tap. Statistical, filtered or otherwise reduced data is retrieved from the local agents by a global manager (the "collect" process).

Caveats

None.

Bugs

Bug fixes, extensions, and other pointers are discussed in the electronic mail forum, bytecounters. To join, send a request to bytecounters-request@venera.isi.edu. Forum exchanges are archived in the file bytecounters/bytecounters.mail, available via anonymous FTP from venera.isi.edu.

Limitations

NNStat presumes a topology of one or more long haul networks gatewayed to local ethernets. A kernel mod required to run with SunOS4. These mods are described in the bytecounters archive.

Hardware Required

Ethernet interface. Sun 3, Sun 4 (SPARC), or PC RT workstation.

Software Required

Distribution is for BSD UNIX, could easily be adapted to any UNIX with promiscuous ethernet support.

Availability

Distribution is available via anonymous FTP from venera.isi.edu, in file pub/NNStat.tar.Z. Documentation is in pub/NNStat.userdoc.ms.Z.

RFC-1147 Network Tools Directory

NPRV -- IP Node/Protocol Reachability Verifier

Keywords

map, routing, status; IP; ping; VMS; free.

Abstract

NPRV is a full-screen, keypad-oriented utility that runs under VAX/VMS. It allows the user to quickly scan through a user-defined list of IP addresses (or domain names) and verify a node's reachability. The node's reachability is determined by performing an ICMP echo, UDP echo and a TCP echo at alternating three second intervals. The total number of packets sent and received are displayed, as well as the minimum, average and maximum round-trip times (in milliseconds) for each type of echo. Additionally, a "trace route" function is performed to determine the path from the local system to the remote host. Once all of the trace route information has filled the screen, a "snapshot" of the screen can be written to a text file. Upon exiting the utility, these text files can be used to generate a logical network map showing host and gateway interconnectivity.

Mechanism

The ICMP echo is performed by sending ICMP ECHO REQUEST packets. The UDP and TCP echoes are performed by connecting to the UDP/TCP echo ports (port number 7). The trace route information is compiled by sending alternating ICMP ECHO REQUEST packets and UDP packets with very large destination UDP port numbers (in two passes). Each packet is initially sent with a TTL (time to live) of 1. This should cause an ICMP TIME EXCEEDED error to be generated by the first routing gateway. Then each packet is sent with a TTL of 2. This should cause an ICMP TIME EXCEEDED error to be generated by the second routing gateway. Then each packet is sent with a TTL of 3, and so on. This process continues until an ICMP ECHO REPLY or UDP PORT UNREACHABLE is received. This indicates that the remote host has been reached and that the trace route information is complete.

Caveats

This utility sends one echo packet per second (ICMP, UDP or TCP), as well as sending out one trace route packet per second. If a transmitted trace route packet is returned in less than one second, another trace route packet is sent in 100 milliseconds. This could cause a significant amount of contention on the local network.

Bugs

None known. Please report any discovered bugs to the author at:

Allen Sturtevant
National Magnetic Fusion Energy Computer Center
Lawrence Livermore National Laboratory
P.O. Box 808; L-561
Livermore, CA 94550
Phone : (415) 422-8266
E-Mail: sturtevant@ccc.nmfecc.gov

Limitations

The user is required to have SYSPRV privilege to per form the ICMP Echo and trace route functions. The utility will still run with this privilege disabled, but only the UDP Echo and TCP Echo information will be displayed. This utility is written in C, but unfortunately it

cannot be easily ported over to UNIX since many VMS system calls are used and all screen I/O is done using the VMS Screen Management Routines.

Hardware Required

Any network interface supported by TGV Incorporated's MultiNet software.

Software Required

VAX/VMS V5.1+ and TGV Incorporated's MultiNet version 2.0.

Availability

For executables only, FTP to the ANONYMOUS account (password GUEST) on CCC.NMFECC.GOV (128.55.128.30) and GET the following files:

[ANONYMOUS.PROGRAMS.NPRV]NPRV.DOC	(ASCII text)
[ANONYMOUS.PROGRAMS.NPRV]NPRV.EXE	(binary)
[ANONYMOUS.PROGRAMS.NPRV]SAMPLE.IPA	(ASCII text)

RFC-1147 Network Tools Directory

nslookup

Keywords

status; DNS; spoof; UNIX, VMS; free.

Abstract

Nslookup is a program used for interactive query of ARPA Internet domain servers. This program is useful for diagnosing routing or mail delivery problems, where often a local domain server is responding with an incorrect internet address. It is essentially a database front end which converts user queries into domain name queries. By default nslookup queries the local domain name server but you can specify additional servers. Additional information beyond the mapping of domain names to internet addresses is possible.

Mechanism

Formats and sends domain name queries.

Caveats

None.

Bugs

None known.

Limitations

None known.

Hardware Required

No restrictions.

Software Required

BSD UNIX or related OS, or VMS.

Availability

Nslookup is part of the "named" distribution, available via anonymous FTP from uunet.uu.net, in directories bsd-sources/src/etc and bsd-sources/src/network, and part of the "bind" distribution, available via anonymous FTP from ucbarpa.berkeley.edu, in directory 4.3. Also available with 4.xBSD UNIX and related operating systems. For VMS, available as part of TGV MultiNet IP software package, as well as Wollongong's WIN/TCP.

RFC-1147 Network Tools Directory

osilog -- OSI event Logger

Keywords

alarm, manager; OSI; UNIX; free.

Abstract

The osilog program receives management event reports for the operation of the ISODE Transport layer (ISO Transport Protocol class 0) on one or more managed systems, formats them suitably to facilitate post processing and records them for future analysis.

Mechanism

It communicates with the System Management Agents (SMAs) on the selected systems via CMIP.

Caveats

The System Management Agent (SMA) must be running on the hosts selected to provide management reports.

Bugs

None known.

Limitations

ISODE Transport Layer only supported by the SMA at present.

Hardware Required

Developed and tested on Sun3.

Software Required

The ISODE protocol suite, BSD UNIX.

Availability

The osilog and related tools, known as OSIMIS (OSI Management Information Service), are publicly available from University College London, England via FTP and FTAM. To obtain information regarding a copy send email to gknight@ac.ucl.cs.uk or call +44 1 380 7366.

RFC-1147 Network Tools Directory

osimic -- OSI Microscope

Keywords

manager, status; OSI; UNIX; free.

Abstract

The osimic program is a human user interface to the management information base on the ISODE Transport layer (ISO Transport Protocol class 0). It allows browsing through the management information tree and enables the manipulation of attribute values. It is implemented using the SunView package of the SunTools window system.

Mechanism

It communicates with the System Management Agent (SMA) on the selected system via CMIP.

Caveats

The System Management Agent (SMA) must be running on the host where the mib is being examined.

Bugs

None known.

Limitations

ISODE Transport Layer only supported by the SMA at present.

Hardware Required

Developed and tested on Sun3.

Software Required

The ISODE protocol suite, BSD UNIX, SunView/SunTools.

Availability

The osimic and related tools, known as OSIMIS (OSI Management Information Service), are publicly available from University College London, England via FTP and FTAM. To obtain information regarding a copy send email to gknight@ac.ucl.cs.uk or call +44 1 380 7366.

RFC-1147 Network Tools Directory

osimon -- OSI Monitor

Keywords

manager, status, traffic; OSI; curses; UNIX; free.

Abstract

The osimon program monitors activity of the ISODE Transport layer (ISO Transport Protocol class 0), displaying entries for the active transport entities and connections. The display is dynamically updated in the case of significant events such as connection opening and closing and packet traffic, as information is received in the form of event reports from a SMA. It uses the UNIX curses package for screen management.

Mechanism

It communicates with the System Management Agent (SMA) on the selected system via CMIP.

Caveats

The System Management Agent (SMA) must be running on the host being monitored.

Bugs

For the terminal type Sun, there are some transient problems with the display.

Limitations

ISODE Transport Layer only supported at present.

Hardware Required

Developed and tested on Sun3 for various terminal types.

Software Required

The ISODE protocol suite, BSD UNIX.

Availability

The osimon and related tools, known as OSIMIS (OSI Management Information Service), are publicly available from University College London, England via FTP and FTAM. To obtain information regarding a copy send email to gknight@ac.ucl.cs.uk or call +44 1 380 7366.

RFC-1147 Network Tools Directory

OSITRACE

Keywords

traffic; OSI; eavesdrop; UNIX; free.

Abstract

OSITRACE is a network performance tool that displays information about ISO TP4 connections. One line of output is displayed for each packet indicating the time, source, destination, length, packet type, sequence number, credit, and any optional parameters contained in the packet. Numerous options are available to control the output of OSITRACE.

To obtain packets to analyze, OSITRACE uses Sun Microsystems' Network Interface Tap (NIT) in SunOS 3.4, 3.5, and 4.0.X. OSITRACE may also obtain data from the NETMON utility which is described as another tool entry.

In Sun systems, OSITRACE may be easily installed: OSI kernel support is not needed, nor is any other form of OSI software support.

Mechanism

This tool has been designed in such a way that code to process different protocol suites may be easily added. As such, OSITRACE also has the ability to trace the DOD TCP protocols.

Caveats

None.

Bugs

Bug reports and questions should be addressed to: ie tools@gateway.mitre.org

Requests to join this mailing list: ie-tools request@gateway.mitre.org

Questions and suggestions can also be directed to: Greg Hollingsworth, gregh@gateway.mitre.org

Limitations

None reported.

Hardware Required

No restriction.

Software Required

SunOS 3.4, 3.5, or 4.0.X, or BSD UNIX-like network protocols with NETMON installed.

Availability

OSITRACE is copyrighted by the MITRE-Washington Networking Center, but freely distributed "as is." It requires retention of a copyright text in code derived from it. The distribution is available by anonymous FTP in pub/pdutrace.tar or pub/pdutrace.tar.Z from aelred-3.ie.org.

RFC-1147 Network Tools Directory

OverVIEW

Keywords

manager, status; IP; NMS, SNMP; DOS.

Abstract

Network and internet monitor; Performance monitor; Fully Graphic user interface; Event logging; TFTP boot server

Mechanism

OverVIEW uses SNMP to query routers, gateways and hosts. Also supports SGMP, PING and is committed to CMIP/CMOT. The SNMP queries allow dynamic determination of configuration and state. Sets of related queries allows monitoring of congestion and faults. The hardware and software are sold as an integrated package.

Caveats

None.

Bugs

None known.

Limitations

256 nodes, 256 nets

Hardware Required

80286, 640K, EGA, mouse.

Software Required

MS-DOS, OverVIEW, Network kernel, Mouse driver, SNMP agents for monitored devices.

Availability

Fully supported product of Proteon, Inc. For more information, contact:

Proteon, Inc.
2 Technology Drive
Westborough, MA 01581 Telex: 928124
Phone: (508) 898-2800
Fax: (508) 366-8901

RFC-1147 Network Tools Directory

ping

Keywords

generator, status; IP; ping; DOS, UNIX, VMS; free.

Abstract

Ping is perhaps the most basic tool for internet management. It verifies that a remote IP implementation and the intervening networks and interfaces are functional. It can be used to measure round trip delay. Numerous versions of the ping program exist.

Mechanism

Ping is based on the ICMP ECHO_REQUEST message.

Caveats

If run repeatedly, ping could generate high system loads.

Bugs

None known.

Limitations

PC/TCP's ping is the only implementation known to support both loose and strict source routing. Though some ping implementations support the ICMP "record route" feature, the usefulness of this option for debugging routes is limited by the fact that many gateways do not correctly implement it.

Hardware Required

No restrictions.

Software Required

None.

Availability

Ping is widely included in TCP/IP distributions. Public domain versions of ping are available via anonymous FTP from uunet.uu.net, in directory bsd sources/src/etc, and from venera.isi.edu, in directory pub.

RFC-1147 Network Tools Directory

proxyd

SNMP proxy agent daemons from SNMP Research.

Keywords

control, status; bridge, ethernet, IP, ring, star; NMS, SNMP; UNIX; library, sourcelib.

Abstract

SNMP proxy agents may be used to permit the monitoring and controlling of network elements which are otherwise not addressable using the SNMP management protocol (e.g., a network bridge that implements a proprietary management protocol). Similarly, SNMP proxy agents may be used to protect SNMP agents from redundant network management agents through the use of caches. Finally, SNMP proxy agents may be used to implement elaborate MIB access policies. The proxy agent daemon listens for SNMP queries and commands from logically remote network management stations, translates and retransmits those as appropriate network management queries or cache lookups, listens for and parses the responses, translates the responses into SNMP responses, and returns those responses as SNMP messages to the network management station that originated the transaction. The proxy agent daemon also emits SNMP traps to identified trap receivers. The proxy agent daemon is architected to make the addition of additional vendor specific variables a straight-forward task. The proxy application comes complete with source code including a powerful set of portable libraries for generating and parsing SNMP messages and a set of command line utili ties.

Mechanism

Network management variables are made available for inspection and/or alteration by means of the Simple Network Management Protocol (SNMP).

Caveats

None.

Bugs

None known.

Limitations

This application is a template for proxy application writers.

Only a few of the many LanBridge 100 variables are supported.

Hardware Required

System from Sun Microsystems, Incorporated.

Software Required

Sun OS 3.5 or 4.x

Availability

This is a commercial product available under license from:

SNMP Research
P.O. Box 8593
Knoxville, TN 37996-4800
(615) 573-1434 (Voice)
(615) 573-9197 (FAX)

Attn: Dr. Jeff Case

RFC-1147 Network Tools Directory

query, ripquery

Keywords

routing; IP; spoof; UNIX; free.

Abstract

Query allows remote viewing of a gateway's routing tables.

Mechanism

Query formats and sends a RIP request or POLL command to a destination gateway.

Caveats

Query is intended to be used a a tool for debugging gateways, not for network management. SNMP is the preferred protocol for network management.

Bugs

None known.

Limitations

The polled gateway must run RIP.

Hardware Required

No restriction.

Software Required

4.3BSD UNIX or related OS.

Availability

Available with routed and gated distributions.

Routed may be obtained via anonymous FTP from uunet.uu.net, in file bsd sources/src/network/routed.tar.Z.

Gated may be obtained via anonymous FTP from devvax.tn.cornell.edu. Distribution files are in directory pub/gated.

RFC-1147 Network Tools Directory

SERAG

Simple Event Reporting and Alarm Generation tool

Keywords

alarm, security; ethernet, IP; NMS, proprietary; UNIX.

Abstract

The Simple Event Reporting and Alarm Generation (SERAG) collects error messages and other event reports from servers on a LAN. Any node with UDP/IP can be the source of such messages/reports. The logging of error messages is integrated with the audit trail facility of the Network Control Server (NCS) from 3COM. Alarms are generated on the NCS based on predefined conditions. Alarms may be sent to the console of the NCS, logged in a file, or routed via WAN to a service center.

SERAG can automatically detect a predefined set of errors in the servers and generate alarms. The breakdown of a server in the LAN may also result in alarm generation.

SERAG creates an error log that can be used for post testing analysis.

Mechanism

The tool searches through the audit trail (error log) files for events specified by the user. The search may be constrained to specific nodes in the network and to a specific time frame. Events may be combined into conditions which are logical expressions (e.g., look for eventA and eventB and not eventC within time frame so and so). This is an interactive query facility to analyze the audit trail (error log).

The user may also ask for such conditions to be checked at regular intervals, and specify routing of error messages in case the condition is satisfied. The checking of such conditions is done by a daemon process running in the background.

Caveats

May impact the performance of the NCS if error logs are big, or if conditions are computationally complex.

Bugs

None known.

Limitations

None reported.

Hardware Required

A workstation running UNIX.

Software Required

Implemented in C (using lex and yacc) on a Sun 3/50.

Also runs under Xenix. Should work with most versions of UNIX.

Availability

Developed jointly by ELAB-RUNIT and Norsk Data:

Tor Didriksen, Ole-Hjalmar Kristensen, Steinar Haug,
Eldfrid Oefsti Oevstedal, Tor Staalhane

ELAB-RUNIT
N-7034 Trondheim
Norway
phone: +47 7 593000
fax : +47 7 532586
email: didrik@idt.unit.no
sthaug@idt.unit.no
kristensen@vax.runit.unit.no

Commercially available from:

Norsk Data A/S
P.O. Box 25, Bogerud
N-0621 Oslo 6
Norway
ref: network management/security management/fault management
phone: +47 2 627500
fax : +47 2 296796

RFC-1147 Network Tools Directory

sma

OSI System Management Agent

Keywords

alarm, manager, status; OSI; UNIX; free.

Abstract

The sma is a CMIP agent which runs on BSD UNIX and provides access to management information on the operation of the ISODE transport layer (ISO Transport Protocol class 0). It also supports the sending of event reports. Activity can be recorded in a log file.

Mechanism

The sma communicates with the active ISODE transport entities using UNIX UDP sockets in order to receive the management information which is made available to other manager processes via CMIP.

Caveats

None.

Bugs

None known.

Limitations

ISODE Transport Layer only supported at present.

Hardware Required

Developed on Sun3, tested on Sun3 and VAXStation.

Software Required

The ISODE protocol suite, BSD UNIX.

Availability

The sma and related tools, known as OSIMIS (OSI Manage ment Information Service), are publicly available from University College London, England via FTP and FTAM. To obtain information regarding a copy send email to gknight@ac.ucl.cs.uk or call +44 1 380 7366.

RFC-1147 Network Tools Directory

Sniffer

Keywords

analyzer, generator, traffic; DECnet, ethernet, IP, NFS, OSI, ring, SMTP, star; eavesdrop; standalone.

Abstract

The Network General Sniffer is a protocol analyzer for performing LAN diagnostics, monitoring, traffic generation, and troubleshooting. The Sniffer protocol analyzer has the capability of capturing every packet on a network and of decoding all seven layers of the OSI protocol model. Capture frame selection is based on several different filters: protocol content at lower levels; node addresses; pattern matching (up to 8 logically-related patterns of 32 bytes each); and destination class. Users may extend the protocol interpretation capability of the Sniffer by writing their own customized protocol interpreters and linking them to the Sniffer software.

The Sniffer displays network traffic information and performance statistics in real time, in user-selectable formats. Numeric station addresses are translated to symbolic names or manufacturer ID names. Network activities measured include frames accepted, Kbytes accepted, and buffer use. Each network version has additional counters for activities specific to that network. Network activity is expressed as frames/second, Kbytes/second, or per cent of network bandwidth utilization.

Data collection by the Sniffer may be output to printer or stored to disk in either print-file or spread-sheet format.

Protocol suites understood by the Sniffer include: Banyan Vines, IBM Token-Ring, Novell Netware, XNS/MS Net (3Com 3+), DECnet, TCP/IP (including SNMP and applications-layer protocols such as FTP, SMTP, and TELNET), X Windows (for X version 11), NFS, and several SUN proprietary protocols (including mount, pmap, RPC, and YP). Supported LANs include: ethernet, Token-ring (4Mb and 16Mb versions), ARCNET, StarLAN, IBM PC Net work (Broadband), and Apple Localtalk Network.

Mechanism

The Sniffer is a self-contained, portable protocol analyzer that require only AC line power and connection to a network to operate. Normally passive (except when in Traffic Generator mode), it captures images of all or of selected frames in a working buffer, ready for immediate analysis and display.

The Sniffer is a standalone device. Two platforms are available: one for use with single network topologies, the other for use with multi-network topologies. Both include Sniffer core software, a modified network interface card (or multiple cards), and optional protocol interpreter suites.

All Sniffer functions may be remotely controlled from a modem-connected PC. Output from the Sniffer can be imported to database or spreadsheet packages.

Caveats

In normal use, the Sniffer is a passive device, and so will not adversely effect network performance. Performance degradation will be observed, of course, if the Sniffer is set to Traffic Generator mode and connected to an active network.

Bugs

None known.

Limitations

None reported.

Hardware Required

None. The Sniffer is a self-contained unit, and includes its own interface card. It installs into a network as would any normal workstation.

Software Required

None.

Availability

The Sniffer is available commercially. For information on your local representative, call or write:

Network General Corporation 4200 Bohannon Drive
Menlo Park, CA 94025 Phone: (415) 688-2700 Fax:
415-321-0855

For acquisition by government agencies, the Sniffer is included on the GSA schedule.

RFC-1147 Network Tools Directory

The SNMP Development Kit

Keywords

manager, status; IP; NMS, SNMP; UNIX; free, sourcelib.

Abstract

The SNMP Development Kit comprises C Language source code for a programming library that facilitates access to the management services of the SNMP (RFC 1098). Sources are also included for a few simple client applications whose main purpose is to illustrate the use of the library. Example client applications query remote SNMP agents in a variety of modes, and generate or collect SNMP traps. Code for an example SNMP agent that supports a subset of the Internet MIB (RFC 1066) is also included.

Mechanism

The Development Kit facilitates development of SNMP based management applications -- both clients and agents. Example applications execute SNMP management operations according to the values of command line arguments.

Caveats

None.

Bugs

Fixed in the next release.

Limitations

None reported.

Hardware Required

The SNMP library source code is highly portable and runs on a wide range of platforms.

Software Required

The SNMP library source code has almost no operating system dependencies and runs in a wide range of environments. Certain portions of the example SNMP agent code are specific to the 4.3BSD implementation of the UNIX system for the DEC MicroVAX.

Availability

The Development Kit is available via anonymous FTP from host allspice.lcs.mit.edu. The copyright for the Development Kit is held by the Massachusetts Institute of Technology, and the Kit is distributed without charge according to the terms set forth in its code and documentation. The distribution takes the form of a UNIX tar file.

Bug reports, questions, suggestions, or complaints may be mailed electronically to snmp-dk@ptt.lcs.mit.edu, although no response in any form is guaranteed. Distribution via UUCP mail may be arranged by contacting the same address. Requests for hard-copy documentation or copies of the distribution on magnetic media are never honored.

RFC-1147 Network Tools Directory

Snmp Libraries and Utilities from SNMP Research.

Keywords

alarm, control, manager, map, routing, status; bridge, DECnet, ethernet, IP, OSI, ring, star; NMS, SNMP; DOS, UNIX, VMS; sourcelib.

Abstract

The SNMP Libraries and Utilities serve two purposes:

1) to act as building blocks for the construction of SNMP-based agent and manager applications; and

2) to act as network management tools for network fire fighting and report generation.

The libraries perform ASN.1 parsing and generation tasks for both network management station applications and network management agent applications. These libraries hide the details of ASN.1 parsing and generation from application writers and make it unnecessary for them to be expert in these areas. The libraries are very robust with considerable error checking designed in. The several command line utilities include applications for retrieving one or many variables, retrieving tables, or effecting commands via the setting of remote network management variables.

Mechanism

The parsing is performed via recursive descent methods. Messages are passed via the Simple Network Management Protocol (SNMP).

Caveats

None.

Bugs

None known.

Limitations

None reported.

Hardware Required

This software has been ported to a wide range of sys tems, too numerous to itemize. It includes worksta tions, general purpose timesharing systems, and embed ded hardware in intelligent network devices such as re peaters, bridges, and routers.

Software Required

C compiler, TCP/IP library from a variety of sources.

Availability

This is a commercial product available under license from:

SNMP Research
P.O. Box 8593
Knoxville, TN 37996-4800
(615) 573-1434 (Voice)
(615) 573-9197 (FAX)
Attn: Dr. Jeff Case

RFC-1147 Network Tools Directory

snmpask

Keywords

manager, status; IP; NMS, SNMP; UNIX.

Abstract

Snmpask is a network monitoring application which gathers specific information from a single network entity at regular intervals and stores this information into UNIX flat files. A report generation package is included in the NYSERNet SNMP Software Distribution to produce reports and graphs from the raw data.

Mechanism

Snmpask uses SNMP to gather its information. The agent which must be queried and the variables to query for are specified in a configuration file.

Caveats

An SNMP agent must be running in the network entity being monitored in order for snmpask to be useful.

Bugs

None outstanding. They are fixed as reports come in. Report bugs to: nysersnmp@nisc.nyser.net

Limitations

SNMP polling is done synchronously. Only a single agent can be polled per snmpask process. Only 16 variables can be requested per snmpask process.

Hardware Required

Developed on Sun 3/60, Sun 3/260, tested on a SPARCstation I, DECstation, and a Solbourne 4/802.

Software Required

Some UNIX variant or some other OS with a Berkeley Socket Compatibility Library.

Availability

Snmpask is available in the NYSERNet SNMP Software Distribution, which is licensed, copyrighted software. To obtain information regarding the package send mail to: snmplisc@nisc.nyser.net or call +1 518-283-8860.

RFC-1147 Network Tools Directory

snmpd (I)

Keywords

manager, status; IP; NMS, SNMP; UNIX.

Abstract

Snmpd is an SNMP agent which runs on UNIX derivatives and answers network management queries from network management stations supporting SNMP. Snmpd also supports the sending of SNMP traps.

Mechanism

Snmpd conforms to SNMP as specified in RFC 1098. Certain user configurable options are manipulated through a simple configuration file.

Caveats

UNIX does not support all of the MIB variables specified in RFC 1066. Snmpd does the best it can to find the answers.

Bugs

None outstanding. They are fixed as reports come in. report bugs to: nysersnmp@nisc.nyser.net

Limitations

See Caveats.

Hardware Required

Developed on Sun 3/60, Sun 3/260, tested on a SPARCsta tion I, DECstation, and a Solbourne 4/802.

Software Required

Some UNIX variant.

Availability

Snmpd is available in the NYSERNet SNMP Software Distribution, which is licensed, copyrighted software. To obtain information regarding the package send mail to: snmplisc@nisc.nyser.net or call +1 518-283-8860.

RFC-1147 Network Tools Directory

snmpd (II)

an SNMP host/gateway agent daemon from SNMP Research.

Keywords

manager, status; bridge, ethernet, IP, ring, star; NMS, SNMP; DOS, UNIX; sourcelib.

Abstract

The snmpd agent daemon listens for and responds to network management queries and commands from logically remote network management stations. The agent daemon also emits SNMP traps to identified trap receivers. The agent daemon is architected to make the addition of additional vendor-specific variables a straight-forward task. The snmpd application comes complete with source code including a powerful set of portable libraries for generating and parsing SNMP messages and a set of command line utilities.

Mechanism

Network management variables are made available for inspection and/or alteration by means of the Simple Network Management Protocol (SNMP).

Caveats

None.

Bugs

None known.

Limitations

Only the operating system variables available without source code modifications to the operating system and device device drivers are supported.

Hardware Required

This software has been ported to a wide range of systems, too numerous to itemize. It includes workstations, general purpose timesharing systems, and embedded hardware in intelligent network devices such asrepeaters, bridges, and routers.

Software Required

C compiler, ".h" files for operating system.

Availability

This is a commercial product available under license from:

SNMP Research
P.O. Box 8593
Knoxville, TN 37996-4800
(615) 573-1434 (Voice)
(615) 573-9197 (FAX)
Attn: Dr. Jeff Case

RFC-1147 Network Tools Directory

snmplookup

Keywords

manager, status; IP; NMS, SNMP; UNIX.

Abstract

Snmplookup is a network monitoring application that allows the interactive querying of a network entity. Snmplookup mimics nslookup, the DNS interactive query tool, in style and feel.

Mechanism

Snmplookup uses SNMP to gather its information. The network entity to be queried and the variable to be retrieved can be entered from the command shell after snmplookup is invoked.

Caveats

An SNMP agent must be running on the network entity being monitored.

Bugs

None outstanding. They are fixed as reports come in. Report bugs to: nysersnmp@nisc.nyser.net

Limitations

See **Caveats.**

Hardware Required

Developed on Sun 3/60, Sun 3/260, tested on a SPARCsta tion I, DECstation, and a Solbourne 4/802.

Software Required

Some UNIX variant or some other OS with a Berkeley Socket Compatibility Library.

Availability

Snmplookup is available in the NYSERNet SNMP Software Distribution, which is licensed, copyrighted software. To obtain information regarding the package send mail to: snmplisc@nisc.nyser.net or call +1 518-283-8860.

RFC-1147 Network Tools Directory

snmpperfmon

Keywords

manager, status; IP; curses, NMS, SNMP; UNIX.

Abstract

Snmpperfmon is a network monitoring application based on the Berkeley curses terminal graphics package and the Simple Network Management Protocol. The application monitors certain interface statistics from a single agent and displays them in tabular form on a standard terminal screen.

Mechanism

Snmpperfmon uses SNMP to gather its information. The agent to be queried is specified on the command line.

Caveats

An SNMP agent must be running in the network entity being monitored in order for snmpperfmon to be useful.

Bugs

None outstanding. They are fixed as reports come in. Report bugs to: nysersnmp@nisc.nyser.net

Limitations

SNMP polling is done synchronously. Only the predeter mined (read "hard coded") interface statistics can be displayed.

Hardware Required

Developed on Sun 3/60, Sun 3/260, tested on a SPARCsta tion I, DECstation, and a Solbourne 4/802.

Software Required

Some UNIX variant or some other OS with a Berkeley Socket Compatibility Library. The "curses" library.

Availability

Snmpperfmon is available in the NYSERNet SNMP Software Distribution, which is licensed, copyrighted software. To obtain information regarding the package send mail to: snmplisc@nisc.nyser.net or call +1 518-283-8860.

RFC-1147 Network Tools Directory

snmppoll

Keywords

manager, status; IP; NMS, SNMP; UNIX.

Abstract

Snmppoll is a network monitoring application which gathers specific information from a network at regular intervals and stores this information into UNIX flat files. A report generation package is included in the NYSERNet SNMP Software Distribution to produce reports and graphs of raw data collected via SNMP.

Mechanism

Snmppoll uses SNMP to gather its information. The agents which must be queried and the variables to query for are specified in a configuration file.

Caveats

An SNMP agent must be running in the network entity being monitored in order for snmppoll to be useful.

Bugs

None outstanding. They are fixed as reports come in. Report bugs to: nysersnmp@nisc.nyser.net

Limitations

SNMP polling is done synchronously.

Hardware Required

Developed on Sun 3/60, Sun 3/260, tested on a SPARCsta tion I, DECstation, and a Solbourne 4/802.

Software Required

Some UNIX variant or some other OS with a Berkeley Socket Compatibility Library.

Availability

Snmppoll is available in the NYSERNet SNMP Software Distribution, which is licensed, copyrighted software. To obtain information regarding the package send mail to: snmplisc@nisc.nyser.net or call +1 518-283-8860.

RFC-1147 Network Tools Directory

snmpquery

Keywords

manager, status; IP; NMS, SNMP; UNIX.

Abstract

Snmpquery is a network monitoring application which allows the simple query of a single network entity from the command line.

Mechanism

Snmpquery uses SNMP to gather its information. The entity to be monitored and the variables to be retrieved must be specified on the command line.

Caveats

An SNMP agent must be running on the network entity being monitored.

Bugs

None outstanding. They are fixed as reports come in. Report bugs to: nysersnmp@nisc.nyser.net

Limitations

Only one network entity can be managed per invocation.

Hardware Required

Developed on Sun 3/60, Sun 3/260, tested on a SPARCsta tion I, DECstation, and a Solbourne 4/802.

Software Required

Some UNIX variant or some other OS with a Berkeley Socket Compatibility Library.

Availability

Snmpquery is available in the NYSERNet SNMP Software Distribution, which is licensed, copyrighted software. To obtain information regarding the package send mail to: snmplisc@nisc.nyser.net or call +1 518-283-8860.

RFC-1147 Network Tools Directory

snmproute

Keywords

manager, routing; IP; NMS, SNMP; UNIX.

Abstract

Snmproute is a network monitoring application that allows the user to query for the entire routing table or a single routing table entry from a network entity.

Mechanism

Snmproute uses SNMP to gather its information. The network entity to be queried and the destination network to be queried for must be specified on the command line.

Caveats

An SNMP agent must be running on the network entity being monitored.

Bugs

None outstanding. They are fixed as reports come in. Report bugs to: nysersnmp@nisc.nyser.net

Limitations

Only one network entity can be queried per invocation.

Hardware Required

Developed on Sun 3/60, Sun 3/260, tested on a SPARCsta tion I, DECstation, and a Solbourne 4/802.

Software Required

Some UNIX variant or some other OS with a Berkeley Socket Compatibility Library.

Availability

Snmproute is available in the NYSERNet SNMP Software Distribution, which is licensed, copyrighted software. To obtain information regarding the package send mail to: snmplisc@nisc.nyser.net or call +1 518-283-8860.

RFC-1147 Network Tools Directory

snmpset

Keywords

control, manager; IP; NMS, SNMP; UNIX.

Abstract

Snmpset is a network management application that allows the alteration of a single variable in a specific agent.

Mechanism

Snmpset uses SNMP to alter the agent variables. The agent to which the set is directed and the variable to alter must be specified on the command line. The user is prompted before any changes are made.

Caveats

An SNMP agent must be running in the network entity being managed in order for snmpset to be useful. In addition, a read-write community must be configured on the agent.

Bugs

None outstanding. They are fixed as reports come in. Report bugs to: nysersnmp@nisc.nyser.net

Limitations

Only one variable can be altered per invocation.

Hardware Required

Developed on Sun 3/60, Sun 3/260, tested on a SPARCsta tion I, DECstation, and a Solbourne 4/802.

Software Required

Some UNIX variant or some other OS with a Berkeley Socket Compatibility Library.

Availability

Snmpset is available in the NYSERNet SNMP Software Distribtion, which is licensed, copyrighted software. To obtain information regarding the package send mail to: snmplisc@nisc.nyser.net or call +1 518-283-8860.

RFC-1147 Network Tools Directory

snmpsrc

Keywords

manager, routing; IP; NMS, SNMP; UNIX.

Abstract

Snmpsrc is a network monitoring application that starts at a specified router in the network and traces the path of a given destination network from the starting router.

Mechanism

Snmpsrc uses SNMP to gather its information. The starting router and destination network must be specified on the command line.

Caveats

An SNMP agent must be running on all of the routers in the path to the destination network in order for a complete path to be reported back to the user. The same SNMP community must also be configured in every SNMP agent in the path to the destination network.

Bugs

None outstanding. They are fixed as reports come in. Report bugs to: nysersnmp@nisc.nyser.net

Limitations

See **Caveats.**

Hardware Required

Developed on Sun 3/60, Sun 3/260, tested on a SPARCstation I, DECstation, and a Solbourne 4/802.

Software Required

Some UNIX variant or some other OS with a Berkeley Socket Compatibility Library.

Availability

Snmpsrc is available in the NYSERNet SNMP Software Distribution, which is licensed, copyrighted software. To obtain information regarding the package send mail to: snmplisc@nisc.nyser.net or call +1 518-283-8860.

RFC-1147 Network Tools Directory

snmpstat

Keywords

manager, status; IP; NMS, SNMP; UNIX.

Abstract

Snmpstat is a network monitoring application that gathers specific information from a network at regular intervals and stores this information into a commercial database. A report generation package is included in the NYSERNet SNMP Software Distribution to produce reports and graphs of raw data collected via SNMP.

Mechanism

Snmpstat uses SNMP to gather its information. The agents which must be queried and the variables to query for are specified in a configuration file.

Caveats

An SNMP agent must be running in the network entity being monitored in order for snmpstat to be useful.

Bugs

None outstanding. They are fixed as reports come in. Report bugs to: nysersnmp@nisc.nyser.net

Limitations

SNMP polling is done synchronously. Currently, Ingres is the only commercial database supported. SQL is the query language being used.

Hardware Required

Developed on Sun 3/60, Sun 3/260, tested on a SPARCsta tion I, DECstation, and a Solbourne 4/802.

Software Required

Some UNIX variant or some other OS with a Berkeley Socket Compatibility Library.

Availability

Snmpstat is available in the NYSERNet SNMP Software Distribution, which is licensed, copyrighted software. To obtain information regarding the package send mail to: snmplisc@nisc.nyser.net or call +1 518-283-8860.

RFC-1147 Network Tools Directory

snmptrapd

Keywords

alarm, manager; IP; NMS, SNMP; UNIX.

Abstract

Snmptrapd is an SNMP trap agent that runs on UNIX derivatives. It receives and logs traps which are generated from snmp agents. A report generation package is included in the NYSERNet SNMP Software Distribution to produce reports and graphs of raw data collected via SNMP.

Mechanism

Snmptrapd conforms to SNMP as specified in RFC 1098. Certain user configurable options are manipulated through a simple configuration file.

Caveats

None.

Bugs

None outstanding. They are fixed as reports come in. Report bugs to: nysersnmp@nisc.nyser.net

Limitations

Snmptrapd only logs traps into a UNIX flat file.

Hardware Required

Developed on Sun 3/60, Sun 3/260, tested on a SPARCsta tion I, DECstation, and a Solbourne 4/802.

Software Required

Some UNIX variant.

Availability

Snmptrapd is available in the NYSERNet SNMP Software Distribution, which is licensed, copyrighted software. To obtain information regarding the package send mail to: snmplisc@nisc.nyser.net or call +1 518-283-8860.

RFC-1147 Network Tools Directory

snmpwatch

Keywords

manager, status; IP; NMS, SNMP; UNIX.

Abstract

Snmpwatch is a network monitoring application that monitors variables in a single network entity and reports when they have changed value.

Mechanism

Snmpwatch uses SNMP to gather its information. The entity to be monitored and the variables to be watched must be specified on the command line. Once a value changes, snmpwatch prints out the value and the variable to the standard output.

Caveats

An SNMP agent must be running on the network entity being monitored. Upon invocation, the initial value of each variable will printed out to the standard output.

Bugs

None outstanding. They are fixed as reports come in. Report bugs to: nysersnmp@nisc.nyser.net

Limitations

Only one network entity can be managed per invocation.

Hardware Required

Developed on Sun 3/60, Sun 3/260, tested on a SPARCsta tion I, DECstation, and a Solbourne 4/802.

Software Required

Some UNIX variant or some other OS with a Berkeley Socket Compatibility Library.

Availability

Snmpwatch is available in the NYSERNet SNMP Software Distribution, which is licensed, copyrighted software. To obtain information regarding the package send mail to: snmplisc@nisc.nyser.net or call +1 518-283-8860.

RFC-1147 Network Tools Directory

snmpxbar

Keywords

manager, status; IP; NMS, SNMP, X; UNIX.

Abstract

Snmpxbar is a network monitoring application based on X-Windows Version 11 Release 2 and the Simple Network Management Protocol. The application monitors a single numeric MIB object and displays its value in a bar chart. Snmpxbar supports color graphics.

Mechanism

Snmpxbar uses SNMP to gather its information. The MIB object to be graphed must be specified on the command line. The polling interval can be changed dynamically from within snmpxbar.

Caveats

An SNMP agent must be running in the network entity being monitored in order for snmpxbar to be useful.

Bugs

Bugs are fixed as reports come in. Report bugs to: nysersnmp@nisc.nyser.net

Limitations

Can only graph one numeric MIB object per invocation.

Hardware Required

Developed on Sun 3/60, Sun 3/260, tested on a SPARCsta tion I, DECstation, and a Solbourne 4/802.

Software Required

Some UNIX variant or some other OS with a Berkeley Socket Compatibility Library. X-Windows.

Availability

Snmpxbar is available in the NYSERNet SNMP Software Distribution, which is licensed, copyrighted software. To obtain information regarding the package send mail to: snmplisc@nisc.nyser.net or call +1 518-283-8860.

RFC-1147 Network Tools Directory

snmpxconn

Keywords

manager, map, status; IP; NMS, SNMP, X; UNIX.

Abstract

Snmpxconn is a network monitoring application based on X-Windows Version 11 Release 2 and the Simple Network Management Protocol. The application monitors a number of (configurable) network entities and graphically depicts the TCP connections associated with the network entities via a TCP topology map.

Mechanism

Snmpxconn uses SNMP to gather its information. A configuration file is used to determine the network entities to be monitored. There are certain command line arguments which manipulate the X environment and SNMP actions.

Caveats

An SNMP agent must be running in the network entity being monitored in order for snmpxconn to be useful.

Bugs

None outstanding. They are fixed as reports come in. Report bugs to: nysersnmp@nisc.nyser.net

Limitations

SNMP polling is done synchronously. The network entities must be configured by manually adding information to a configuration file.

Hardware Required

Developed on Sun 3/60, Sun 3/260, tested on a SPARCsta tion I, DECstation, and a Solbourne 4/802.

Software Required

Some UNIX variant or some other OS with a Berkeley Socket Compatibility Library. X-Windows.

Availability

Snmpxconn is available in the NYSERNet SNMP Software Distribution, which is licensed, copyrighted software. To obtain information regarding the package send mail to: snmplisc@nisc.nyser.net or call +1 518-283-8860.

RFC-1147 Network Tools Directory

snmpxmon

Keywords

manager, map, status; IP; NMS, SNMP, X; UNIX.

Abstract

Snmpxmon is a network monitoring application based on X-Windows Version 11 Release 2 and the Simple Network Management Protocol. This application will determine the status of sites and links it is configured to monitor (via its configuration file) by querying the designated sites and then displaying the result in a map form. Snmpxmon supports color graphics.

Mechanism

Snmpxmon uses SNMP to gather its information. A configuration file is used to design the topology map. There are certain command line arguments which manipulate the X environment and SNMP actions.

Caveats

An SNMP agent must be running in the network entity being monitored in order for snmpxmon to be useful.

Bugs

None outstanding. They are fixed as reports come in. Report bugs to: nysersnmp@nisc.nyser.net

Limitations

SNMP polling is done synchronously. The topology map must be configured by hand.

Hardware Required

Developed on Sun 3/60, Sun 3/260, tested on a SPARCsta tion I, DECstation, and a Solbourne 4/802.

Software Required

Some UNIX variant or some other OS with a Berkeley Socket Compatibility Library. X-Windows.

Availability

Snmpxmon is available in the NYSERNet SNMP Software Distribution, which is licensed, copyrighted software. To obtain information regarding the package send mail to: snmplisc@nisc.nyser.net or call +1 518-283-8860.

RFC-1147 Network Tools Directory

snmpxperf

Keywords

manager, status; IP; NMS, SNMP, X; UNIX.

Abstract

Snmpxperf is a network monitoring application based on X-Windows Version 11 Release 2 and the Simple Network Management Protocol. The application monitors a single numeric MIB object and displays its value in an EKG style histogram. Snmpxperf supports color graphics.

Mechanism

Snmpxperf uses SNMP to gather its information. The MIB object to be graphed must be specified on the command line. The polling interval can be changed dynamically from within snmpxperf.

Caveats

An SNMP agent must be running in the network entity being monitored in order for snmpxperf to be useful.

Bugs

Auto-scaling sometimes doesn't downscale the EKG-graph enough on large spikes. This results in some of the graph running into the button boxes at the top of the window. Generally, Bugs are fixed as reports come in. Report bugs to: nysersnmp@nisc.nyser.net

Limitations

Can only graph one numeric MIB object per invocation.

Hardware Required

Developed on Sun 3/60, Sun 3/260, tested on a SPARCsta tion I, DECstation, and a Solbourne 4/802.

Software Required

Some UNIX variant or some other OS with a Berkeley Socket Compatibility Library. X-Windows.

Availability

Snmpxperf is available in the NYSERNet SNMP Software Distribution, which is licensed, copyrighted software. To obtain information regarding the package send mail to: snmplisc@nisc.nyser.net or call +1 518-283-8860.

RFC-1147 Network Tools Directory

snmpxperfmon

Keywords

manager, status, traffic; IP; NMS, SNMP, X; UNIX.

Abstract

Snmpxperfmon is a network monitoring application based on X-Windows Version 11 Release 2 and the Simple Network Management Protocol. The application monitors a single Network Entity and displays graphical information pertaining to the entities interface traffic statistics. Snmpxperfmon supports color graphics.

Mechanism

Snmpxperfmon uses SNMP to gather its information. The MIB agent to be polled must be specified on the command line. The agent is then queried about all of its interfaces. Four EKG-style graphs are constructed for each interface (input pkts, output pkts, input Octets, output Octets).

Caveats

An SNMP agent must be running in the network entity being monitored in order for snmpxperfmon to be useful.

Bugs

Generally, bugs are fixed as reports come in. Report bugs to: nysersnmp@nisc.nyser.net

Limitations

Can only graph one network entity per invocation. Can only graph the amount of interfaces which will fit on a single bitmap display. Does not auto-scale or resize.

Hardware Required

Developed on Sun 3/60, Sun 3/260, tested on a SPARCsta tion I, DECstation, and a Solbourne 4/802.

Software Required

Some UNIX variant or some other OS with a Berkeley Socket Compatibility Library. X-Windows.

Availability

Snmpxperfmon is available in the NYSERNet SNMP Software Distribution, which is licensed, copyrighted software. To obtain information regarding the package send mail to: snmplisc@nisc.nyser.net or call +1 518-283-8860.

RFC-1147 Network Tools Directory

snmpxrtmetric

Keywords

manager, routing; IP; NMS, SNMP, X; UNIX.

Abstract

Snmpxrtmetric is a network monitoring application based on X-Windows Version 11 Release 2 and the Simple Net work Management Protocol. The application monitors the routing table of a specific agent and displays the RIP routing metric of certain destination networks in bar chart format.

Mechanism

Snmpxrtmetric uses SNMP to gather its information. A configuration file is used to determine which destination networks will be graphed. The agent to be queried is specified on the command line. Snmpxrtmetrtic supports color graphics.

Caveats

An SNMP agent must be running in the network entity being monitored in order for snmpxrtmetric to be use ful.

Bugs

None outstanding. They are fixed as reports come in. Report bugs to: nysersnmp@nisc.nyser.net

Limitations

SNMP polling is done synchronously. The destination networks must be configured by manually adding information to a configuration file.

Hardware Required

Developed on Sun 3/60, Sun 3/260, tested on a SPARCstation I, DECstation, and a Solbourne 4/802.

Software Required

Some UNIX variant or some other OS with a Berkeley Socket Compatibility Library. The X window system.

Availability

Snmpxrtmetric is available in the NYSERNet SNMP Software Distribution, which is licensed, copyrighted software. To obtain information regarding the package send mail to: snmplisc@nisc.nyser.net or call +1 518 283-8860.

RFC-1147 Network Tools Directory

SpiderMonitor P220, K220 and SpiderAnalyzer P320, K320

Keywords

alarm, analyzer, generator, traffic; DECnet, ethernet, IP, OSI; eavesdrop; standalone; sourcelib.

Abstract

The SpiderMonitor and SpiderAnalyzer are protocol analyzers for performing ethernet LAN diagnostics, monitoring, traffic generation, and troubleshooting. The SpiderMonitor has the capability of capturing every packet on a network and of decoding the first four layers of the OSI protocol model. The SpiderAnalyzer has additional software for decoding higher protocol layers. Protocol suites understood: TCP/IP (including SNMP and applications-layer protocols), OSI, XNS, DECnet and IPX. User-definable decodes can be written in 'C' with the Microsoft version 5.0 'C' compiler. A decode guide is provided.

The SpiderAnalyzer supports multiple simultaneous filters for capturing packets using predefined patterns and error states. Filter patterns can also trigger on NOT matching 1 or more filters, an alarm, or a specified time.

The SpiderAnalyzer can also employ TDR (Time Domain Reflectometry) to find media faults, open or short circuits, or transceiver faults. It can transmit OSI, XNS, and Xerox link-level echo packets to user specified stations, performs loop round tests.

In traffic generation mode, the SpiderAnalyzer has the ability to generate packets at random intervals of random lengths or any combination of random or fixed interval or length, generation of packets with CRC errors, or packets that are too short, or packets that are too long.

Output from the SpiderMonitor/Analyzer can be imported to database or spreadsheet packages.

Mechanism

The SpiderMonitor and Spider Analyzer are available as stand-alone, IBM PC compatible packages based upon a Compaq III portable system, or as a plug-in boards for any IBM XT/AT compatible machine. The model 220 (SpiderMonitor) systems provide a functional base suited for most network management needs. The model 320 (SpiderAnalyzer) systems provide extended functionality in the development mode and traffic generation mode as well more filtering capabilities than the 220 models.

Caveats

Traffic generation will congest an operational ether net.

Bugs

None known.

Limitations

Monitoring of up to 1024 stations and buffering of up to 1500 packets. The model 220 provides for 3 filters with a filter depth of 46 bytes. The model 320 pro vides for 4 filters and a second level of filtering with a filter depth of 64 bytes.

Hardware Required

PX20s are self contained, the KX20s require an IBM PC/XT-AT compatible machine with 5

megabytes of hard disk storage and the spare slot into which the board kit is plugged.

Software Required

None. The SpiderAnalyzer requires the Microsoft 'C' Compiler, Version 5.0 for writing user defined decodes.

Availability

The SpiderMonitor/Analyzer is available commercially. For information on your local representative, call or write:

Spider Systems, Inc.
12 New England Executive Park
Burlington, MA 01803
Telephone: 617-270-3510
FAX: 617-270-9818

RFC-1147 Network Tools Directory

SPIMS -- the Swedish Institute of Computer Science (SICS) Protocol Implementation Measurement System tool.

Keywords

benchmark, debugger; IP, OSI; spoof; UNIX.

Abstract

SPIMS is used to measure the performance of protocol and "protocol-like" services including response time (two-way delay), throughput and the time to open and close connections. It has been used to:

- benchmark alternative protocol implementations,
- observe how performance varies when parameters in specific implementations have been varied (i.e., to tune parameters).

SPIMS currently has interfaces to the DoD Internet Protocols: UDP, TCP, FTP, SunRPC, the OSI protocols from the ISODE 4.0 distribution package: FTAM, ROSE, ISO TP0 and to Sunlink 5.2 ISO TP4 as well as Stanford's VMTP. Also available are a rudimentary set of benchmarks, stubs for new protocol interfaces and a user manual. For an example of the use of SPIMS to tune protocols, see:

> Nordmark & Cheriton, "Experiences from VMTP: How to achieve low response time," IFIP WG6.1/6.4:Protocols for High-Speed Networks, May 1989, Zurich. To be published.

Mechanism

SPIMS runs as user processes and uses a TCP connection for measurement set-up. Measurements take place between processes over the measured protocol. SPIMS generates messages and transfers them via the measured protocol service according to a user-supplied specification. SPIMS has a unique measurement specification language that is used to specify a measurement session. In the language there are constructs for different application types (e.g., bulk data transfer), for specifying frequency and sequence of messages, for distribution over message sizes and for combining basic specifications. These specifications are independent of both protocols and protocol implementations and can be used for benchmarking. For more details on the internals of SPIMS, see: Nordmark & Gunningberg, "SPIMS: A Tool for Protocol Implementation Performance Measurements" Proc. of 13:th Conf. on Local Computer Networks, Minneapolis 1989, pp 222-229.

Caveats

None.

Bugs

None known.

Limitations

None reported.

Hardware Required

No restrictions.

Software Required

SPIMS is implemented on UNIX, including SunOS 4., 4.3BSD UNIX, DN (UNIX System V, with extensions) and Ultrix 2.0/3.0. It requires a TCP connection for measurement set-up. No kernel modifications or any modifications to measured protocols are required.

Availability

SPIMS is not in the public domain; the software is covered by licenses. The Swedish Institute of Computer Science has released the research prototype of SPIMS for research and non-commercial use. Commercial organizations may obtain the research prototype, but it is for internal research only and for no commercial use whatsoever. A commercial, supported version of SPIMS is distributed by TeleLOGIC Uppsala AB, Sweden.

For universities and non-profit organizations, SPIMS source code is distributed free of charge. There are two ways to get the software:

1. FTP. If you have an Internet FTP connection, you can use anonymous FTP to sics.se [192.16.123.90], and retrieve the file in pub/spimsdist/dist890915.tar.Z (this is a .6MB tar image) in BINARY mode. Log in as user anonymous and at the password prompt, use your complete electronic mail address.

2. On a Sun 1/4-inch cartridge tape. For mailing, a handling fee of US$150.00 will be charged. Submit a bank check with the request. Do not send tapes or envelopes.

For other organizations, the SPIMS source code for the research prototype is distributed for a one-time fee of US$500.00. Organizations interested in the research prototype need to contact SICS via email and briefly motivate why they qualify (non-commercial use) for the research prototype. They will thereafter get a permission to obtain a copy from the same distribution source as for universities.

For more information about the research prototype distribution, contact:

Swedish Institute of Computer Science
Att: Birgitta Klingenberg
P.O. Box 1263
S-164 28 Kista
SWEDEN
e-address: spims@sics.se
Phone: +46-8-7521500, Fax: +46-8-7517230

TeleLOGIC Uppsala AB, a subsidiary of Swedish Telecom, distributes and supports a version of SPIMS for commercial use. It consists of object code for SunOS 4., 4.3BSD UNIX, DNIX, and Ultrix 2.0/3.0. Support for other UNIX-like implementations will be considered according to demand. The same interfaces to the DoD Internet and OSI protocols from the ISODE 4.0 are included as well as a user manual.

For further information about SPIMS for the commercial user please contact:

Claes Hojenberg
TeleLOGIC Uppsala AB
P.O. Box 1218
S-751 42 UPPSALA
Sweden
e-address: claes@uplog.se
Phone: +46-18-189400, Fax: +46-18-132039

RFC-1147 Network Tools Directory

spray

Keywords

benchmark, generator; IP; ping; UNIX.

Abstract

Spray is a traffic generation tool that generates RPC or UDP packets, or ICMP Echo Requests. The packets are sent to a remote procedure call application at the destination host. The count of received packets is retrieved from the remote application after a certain number of packets have been transmitted. The difference in packets received versus packets sent represents (on a LAN) the packets that the destination host had to drop due to increasing queue length. A measure of throughput relative to system speed and network load can thus be obtained.

Mechanism

See above.

Caveats

Spray can congest a network.

Bugs

None known.

Limitations

None reported.

Hardware Required

No restrictions.

Software Required

SunOS

Availability

Supplied with SunOS.

RFC-1147 Network Tools Directory

tcpdump

Keywords

traffic; ethernet, IP, NFS; UNIX, VMS; free.

Abstract

Tcpdump can interpret and print headers for the following protocols: ethernet, IP, ICMP, TCP, UDP, NFS, ND, ARP/RARP, AppleTalk. Tcpdump has proven useful for examining and evaluating the retransmission and window management operations of TCP implementations.

Mechanism

Much like etherfind, tcpdump writes a log file of the frames traversing an ethernet interface. Each output line includes the time a packet is received, the type of packet, and various values from its header.

Caveats

None.

Bugs

None known.

Limitations

Public domain version requires a kernel patch for SunOS.

Hardware Required

Ethernet.

Software Required

BSD UNIX or related OS, or VMS.

Availability

Available, though subject to copyright restrictions, via anonymous FTP from ftp.ee.lbl.gov. The source and documentation for the tool is in compressed tar format, in file tcpdump.tar.Z. Also available from spam.itstd.sri.com, in directory pub. For VMS hosts with DEC ethernet controllers, available as part of TGV MultiNet IP software package.

RFC-1147 Network Tools Directory

tcplogger

Keywords

traffic; IP; eavesdrop; UNIX; free.

Abstract

Tcplogger consists of modifications to the 4.3BSD UNIX source code, and a large library of post-processing software. Tcplogger records timestamped information from TCP and IP packets that are sent and received on a specified connection. For each TCP packet, information such as sequence number, acknowledgement sequence number, packet size, and header flags is recorded. For an IP packet, header length, packet length and TTL values are recorded. Customized use of the TCP option field allows the detection of lost or duplicate pack ets.

Mechanism

Routines of 4.3BSD UNIX in the netinet directory have been modified to append information to a log in memory. The log is read continuously by a user process and written to a file. A TCP option has been added to start the logging of a connection. Lots of post processing software has been written to analyze the data.

Caveats

None.

Bugs

None known.

Limitations

To get a log at both ends of the connection, the modified kernel should be run at both the hosts. All connections are logged in a single file, but software is provided to filter out the record of a single connection.

Hardware Required

No restrictions.

Software Required

4.3BSD UNIX (as modified for this tool).

Availability

Free, although a 4.3BSD license is required. Contact Olafur Gudmundsson (ogud@cs.umd.edu).

RFC-1147 Network Tools Directory

TokenVIEW

Keywords

control, manager, status; ring; NMS, proprietary; DOS.

Abstract

Network Management tool for 4/16 Mbit IEEE 802.5 Token Ring Networks. Monitors active nodes and ring errors. Maintains database of nodes, wire centers and their connections. Separate network management ring allows remote configuration of wire centers.

Mechanism

A separate network management ring used with Proteon Intelligent Wire Centers allows wire center configuration information to be read and modified from a single remote workstation. A log of network events used with a database contain nodes, wire centers and their connections, facilitates tracking and correction of network errors. Requires an "E" series PROM, sold with package.

Caveats

Currently, only ISA bus cards support the required E series PROM.

Bugs

None known.

Limitations

256 nodes, 1 net.

Hardware Required

512K RAM, CGA or better, hard disk, mouse supported.

Software Required

MS-DOS, optional mouse driver

Availability

Fully supported product of Proteon, Inc. Previously sold as Advanced Network Manager (ANM). For more infomation, contact:

Proteon, Inc.
2 Technology Drive
Westborough, MA 01581
Phone: (508) 898-2800
Fax: (508) 366-8901
Telex: 928124

RFC-1147 Network Tools Directory

traceroute

Keywords

routing; IP; ping; UNIX, VMS; free.

Abstract

Traceroute is a tool that allows the route taken by packets from source to destination to be discovered. It can be used for situations where the IP record route option would fail, such as intermediate gateways discarding packets, routes that exceed the capacity of an datagram, or intermediate IP implementations that don't support record route. Round trip delays between the source and intermediate gateways are also reported allowing the determination of individual gateways contribution to end-to-end delay.

Enhanced versions of traceroute have been developed that allow specification of loose source routes for datagrams. This allows one to investigate the return path from remote machines back to the local host.

Mechanism

Traceroute relies on the ICMP TIME_EXCEEDED error reporting mechanism. When an IP packet is received by an gateway with a time-to-live value of 0, an ICMP packet is sent to the host which generated the packet. By sending packets to a destination with a TTL of 0, the next hop can be identified as the source of the ICMP TIME EXCEEDED message. By incrementing the TTL field the subsequent hops can be identified. Each packet sent out is also time stamped. The time stamp is returned as part of the ICMP packet so a round trip delay can be calculated.

Caveats

Some IP implementations forward packets with a TTL of 0, thus escaping identification. Others use the TTL field in the arriving packet as the TTL for the ICMP error reply, which delays identification.

Sending datagrams with the source route option will cause some gateways to crash. It is considered poor form to repeat this behavior.

Bugs

None known.

Limitations

Most versions of UNIX have errors in the raw IP code that require kernel mods for the standard version of traceroute to work. A version of traceroute exists that runs without kernel mods under SunOS 3.5 (see below), but it only operates over an ethernet inter face.

Hardware Required

No restrictions.

Software Required

BSD UNIX or related OS, or VMS.

Availability

Available by anonymous FTP from ftp.ee.lbl.gov, in file traceroute.tar.Z. It is also available from uc.msc.umn.edu.

A version of traceroute that supports Loose Source Record Route, along with the source code of the required kernel modifications and a Makefile for installing them, is available via anonymous FTP from zerkalo.harvard.edu, in directory pub, file traceroute_pkg.tar.Z.

A version of traceroute that runs under SunOS 3.5 and does NOT require kernel mods is available via anonymous FTP from dopey.cs.unc.edu, in file ~ftp/pub/traceroute.tar.Z.

For VMS, traceroute is available as part of TGV MultiNet IP software package.

RFC-1147 Network Tools Directory

TRPT -- transliterate protocol trace

Keywords

traffic; IP; eavesdrop; UNIX; free.

Abstract

TRPT displays a trace of a TCP socket events. When no options are supplied, TRPT prints all the trace records found in a system, grouped according to TCP connection protocol control block (PCB).

An example of TRPT output is:

```
38241 ESTABLISHED:input
[e0531003..e0531203)@6cc5b402(win=4000)<ACK> -> ESTA BLISHED
38241 ESTABLISHED:user RCVD -> ESTABLISHED
38266 ESTABLISHED:output
6cc5b402@e0531203(win=4000)<ACK> -> ESTABLISHED
38331 ESTABLISHED:input
[e0531203..e0531403)@6cc5b402(win=4000)<ACK,FIN,PUSH>
-> CLOSE_WAIT
38331 CLOSE_WAIT:output
6cc5b402@e0531404(win=3dff)<ACK> -> CLOSE_WAIT
38331 CLOSE_WAIT:user RCVD -> CLOSE_WAIT
38343 LAST_ACK:output
6cc5b402@e0531404(win=4000)<ACK,FIN> -> LAST_ACK
38343 CLOSE_WAIT:user DISCONNECT -> LAST_ACK
38343 LAST_ACK:user DETACH -> LAST_ACK
```

Mechanism

TRPT interrogates the buffer of TCP trace records that is created when a TCP socket is marked for debugging.

Caveats

Prior to using TRPT, an analyst should take steps to isolate the problem connection and find the address of its protocol control blocks.

Bugs

None reported.

Limitations

A socket must have the debugging option set for TRPT to operate. Another problem is that the output format of TRPT is difficult.

Hardware Required

No restrictions.

Software Required

BSD UNIX or related OS.

Availability

Included with BSD and SunOS distributions. Available via anonymous FTP from

uunet.uu.net, in file bsdsources/src/etc/trpt.tar.Z.

RFC-1147 Network Tools Directory

TTCP

Keywords

benchmark, generator; IP; ping; UNIX, VMS; free.

Abstract

TTCP is a traffic generator that can be used for testing end-to-end throughput. It is good for evaluating TCP/IP implementations.

Mechanism

Cooperating processes are started on two hosts. The open a TCP connection and transfer a high volume of data. Delay and throughput are calculated.

Caveats

Will greatly increase system load.

Bugs

None known.

Limitations

None reported.

Hardware Required

No restrictions.

Software Required

BSD UNIX or related OS, or VMS.

Availability

Source for BSD UNIX is available via anonymous FTP from vgr.brl.mil, in file ftp/pub/ttcp.c, and from sgi.com, in file sgi/src/ttcp.c. A version of TTCP has also been submitted to the USENET news group comp.sources.unix. For VMS, ttcp.c is included in the MultiNet Programmer's Kit, a standard feature of TGV MultiNet IP software package.

RFC-1147 Network Tools Directory

Unisys Network Control Center (NCC)

Keywords

alarm, benchmark, control, generator, manager, map, reference, status, traffic; ethernet, FDDI, IP; NMS, ping, SNMP; UNIX.

Abstract

The Unisys Defense Systems Network Control Center (NCC) provides high-performance software to support the management and control of TCP/IP-based networks. The network management system uses the Simple Network Management Protocol (SNMP) to exchange management information between the NCC and network devices. The NCC supports the Management Information Base (MIB) [RFC-1066] and the Structure and Identification of Management Information for TCP/IP-based Internets [RFC-1065]. In addition, Unisys has extended the MIB definitions to support the features of Unisys FDDI LAN devices, such as the FDDI Smart Concentrators, the FDDI Host Network Front Ends, and the Remote FDDI, FDDI-to-LAN, and FDDI-to-DDN gateways. The NCC supports seven applications. The network topology map displays the physical and logical maps of the network. The configuration management tool supports the modification and validation of network device configuration data as well as the modification of MIB configuration data. The performance monitoring tool supports the collection and analysis of statistical parameters from network devices. The status monitoring tool reports on the up/down status and responsiveness of network devices using ICMP. The accounting tool is used to collect, store, and display user job activity at the subscriber hosts. The NCC database entry sup ports RFC 1066 object definitions and Unisys-specific object definitions to support the Unisys FDDI devices. And finally, the trap reporting tool reports the arrival of error and event notifications using UDP datagrams. The NCC supports all the trap messages defined in RFC 1098.

Mechanism

The NCC is based on the Simple Network Management Pro tocol (SNMP).

Caveats

None.

Bugs

None known.

Limitations

None reported.

Hardware Required

A minimal platform consists of a Sun 3/60FC-8, with at least 200 MB disk and cartridge tape (1/4"). A full sized color monitor, more disk, and a workstation based on a higher performance processor is beneficial to NCC activities.

Software Required

SunOS Version 4.0 running the SunView windowing environment and the SYBASE Relational Data Base Manage ment System.

Availability

Commercially available as a turn-key package or as a software product from:

Unisys Defense Systems

5151 Camino Ruiz
Camarillo, California 93010
(805) 987-6811
(Dale Russell <dsr@cam.unisys.com>)

RFC-1147 Network Tools Directory

WIN/MGT Station -- Network Management Station for SunOS.

Keywords

alarm, control, manager, routing, status, traffic; ethernet, IP; NMS, SNMP, X; UNIX; library.

Abstract

WIN/MGT Station for SunOS is a network management software product based on the SNMP. It provides the capability to manage standards-based networking products from The Wollongong Group as well as other ven dors. Fully compliant with RFCs 1065, 1066 and 1098, WIN/MGT Station uses a menu-driven graphical user interface.

WIN/MGT capabilities include configuration, performance and fault management for SNMP-based agents. The WIN/MGT station can perform polling to monitor the status of all MIB variables defined in RFC 1066, "Management Information Base for network management of TCP/IP-based internets." In addition, the WIN/MGT Station can process "trap" messages from SNMP agents. Furthermore, the WIN/MGT Station can support any private extension to the Management Information Base with minimal user configuration.

An icon-driven network interface map allows the user to monitor their network topology and status. Changes in the operational status of any manageable network element is displayed visually and audibly.

The WIN/MGT package includes an Applications Programming Interface (API) for the "C" language. The API is a set of libraries that enable an applications program to perform SNMP "set" and "get" operations. This allows users to integrate site-specific applications with WIN/MGT.

SNMP agent software for the Sun 3 host is also provided so that the Network Management Station itself can also be monitored and managed.

Mechanism

The WIN/MGT Station uses SNMP to monitor and control SNMP agents.

Caveats

None.

Bugs

None known.

Limitations

A theoretical limitation of approximately 18,000 network elements can be managed.

Hardware Required

Any model of Sun 3 system. Recommended minimums include 8 MB RAM, 100 MB disk space (30 MB to start), and color monitor. Also tested on DECstation 3100, PS/2 (with SCO UNIX) and Macintosh IIcx computer using A/UX.

Software Required

SunOS 4.x. MIT X Window System, Release 11, version 3, or OpenWindows (X.11/NeWS) from Sun Microsystems, Inc. WIN/MGT Station for SunOS is provided on 1/4" tape in cpio format.

Availability

A commercial product of:

The Wollongong Group, Inc.
1129 San Antonio Rd.
Palo Alto, CA 94303
(415) 962-7200 br fax (415) 968-3619
internet oldera@twg.com

RFC-1147 Network Tools Directory

xnetmon, xpmon

Keywords

alarm, manager, map, status; IP; NMS, SNMP; UNIX.

Abstract

Xnetmon and xpmon provide graphical representation of performance and status of SNMP-capable network ele ments. Xnetmon presents a schematic network map representing the up/down status of network elements; xpmon draws a pen plot style graph of the change over time of any arbitrary MIB object (RFC1066). Both xnetmon and xpmon use the SNMP (RFC1098) for retrieving status and performance data.

Mechanism

Xnetmon polls network elements for the status of their interfaces on a controllable polling interval. Pop-up windows displaying the values of any MIB variable are supported by separate polls. When SNMP traps are received from a network element, that element and all adjacent elements are immediately re-polled to update their status. The layout of the network map is statically configured. Xpmon repeatedly polls (using SNMP) the designated network element for the value of the designated MIB variable on the user-specified interval. The change in the variable is then plotted on the strip chart. The strip chart regularly adjusts its scale to the current maximum value on the graph.

Caveats

Polling intervals should be chosen with care so as not to affect system performance adversely.

Bugs

None known.

Limitations

None reported.

Hardware Required

Distributed and supported for Sun-3 systems.

Software Required

SunOS 3.5 or 4.x; X11, release 2 or 3.

Availability

Commercial product of:

Wellfleet Communications, Inc.
12 DeAngelo Drive
Bedford, MA 01730-2204
(617) 275-2400

RFC-1147 Network Tools Directory

XNETMON
an X windows based SNMP network management station
from SNMP Research.

Keywords

alarm, control, manager, map, routing, security, status; DECnet, ethernet, IP, OSI, ring, star; NMS, SNMP, X; DOS, UNIX, VMS; sourcelib.

Abstract

The XNETMON application implements a powerful network management station based on the X window system. It provides network managers tools for fault management, configuration management, performance management, and security management. It can be successfully used with many types of networks including those based on various LAN media, and wide area networks. XNETMON has been used with multiprotocol devices including those which support TCP/IP, DECnet, and OSI protocols. The fault management tool displays the map of the network configuration with node and link state indicated in one of several colors to indicate current status. Alarms may be enabled to alert the operator of events occurring in the network. Events are logged to disk. The configuration management tool may be used to edit the network management information base stored in the network management station to reflect changes occurring in the network. Other features include graphs and tabular tools for use in fault and performance management and mechanisms by which additional variables, such as vendor-specific variables, may be added. The XNETMON application comes complete with source code including a powerful set of portable libraries for generating and parsing SNMP messages. Output data from XNETMON may be transferred via flat files for additional report generation by a variety of statistical packages.

Mechanism

The XNETMON application is based on the Simple Network Management Protocol (SNMP). Polling is performed via the powerful SNMP get-next operator and the SNMP get operator. Trap directed polling is used to regulate the focus and intensity of the polling.

Caveats

None.

Bugs

None known.

Limitations

The monitored and managed nodes must implement the SNMP over UDP per RFC 1098 or must be reachable via a proxy agent.

Hardware Required

X windows workstation with UDP socket library. Monochrome is acceptable but color is far superior.

Software Required

X windows version 11 release 3 or later.

Availability

This is a commercial product available under license from:

SNMP Research
P.O. Box 8593
Knoxville, TN 37996-4800
(615) 573-1434 (Voice)
(615) 573-9197 (FAX)
Attn: Dr. Jeff Case

RFC-1147 Network Tools Directory

xnetperfmon
a graphical network performance and fault management tool
from SNMP Research.

Keywords

manager, status; DECnet, ethernet, IP, OSI, ring, star; NMS, SNMP, X; DOS, UNIX, VMS; sourcelib.

Abstract

Xnetperfmon may be used to plot SNMP variables as a graphical display. These graphs are often useful for fault and performance management. Variables may be plotted as gauges versus time. Alternatively, counters may be plotted as delta count/delta time (rates). The user may easily customize the variables to be plotted, labels, step size, update interval, and the like. The scales automatically adjust whenever a point to be plotted would go off scale.

Mechanism

The xnetperfmon application communicates with remote agents or proxy agents via the Simple Network Manage ment Protocol (SNMP).

Caveats

All plots for a single invocation of xnetperfmon must be for variables provided by a single network management agent. However, multiple invocations of xnetperfmon may be active on a single display simultaneously or proxy agents may be used to summarize information at a common point.

Bugs

None known.

Limitations

None reported.

Hardware Required

Systems supporting X windows.

Software Required

X Version 11 release 2 or later.

Availability

This is a commercial product available under license from:

SNMP Research
P.O. Box 8593
Knoxville, TN 37996-4800
(615) 573-1434 (Voice)
(615) 573-9197 (FAX)
Attn: Dr. Jeff Case

RFC-1147 Network Tools Directory

xup

Keywords

status; ping, X; HP.

Abstract

Xup uses the X-Windows to display the status of an "interesting" set of hosts.

Mechanism

Xup uses ping to determine host status.

Caveats

Polling for status increases network load.

Bugs

None known.

Limitations

None reported.

Hardware Required

Runs only on HP series 300 and 800 workstations.

Software Required

Version 10 of X-Windows.

Availability

A standard command for the HP 300 & 800 Workstations.

RFC-1147 Network Tools Directory

Network Management Tutorial

RFC-1147 Network Tools Directory - Network Management Tutorial

Introduction

This tutorial is an overview of the practice of network management. Reading this section is no substitute for knowing your system, and knowing how it is used. Do not wait until things break to learn what they ought to do or how they usually work: a crisis is not the time for determining how "normal" packet traces should look. Furthermore, it takes little imagination to realize that you do not want to be digging through manuals while your boss is screaming for network service to be restored.

We assume an acquaintance with the TCP/IP protocol suite and the Internet architecture. There are many available references on these topics, several of which are listed below in Section 7.

Since many of the details of network management are systemspecific, this tutorial is a bit superficial. There is, however, a more fundamental problem in prescribing network management practices: network management is not a well- understood endeavor. At present, the cutting edge of network management is the use of distributed systems to collect and exchange status information, and then to display the data as histograms or trend lines. It is not clear that we know what data should be collected, how to analyze it when we get it, or how to structure our collection systems. For now, automated, real-time control of internets is an aspiration, rather than a reality. The communications systems that we field are apparently more complex than we can comprehend, which no doubt accounts in part for their fre- quently surprising behavior.

The first section of this tutorial lists the overall goals and functions of network management. It presents several aspects of network management, including system monitoring, fault detection and isolation, performance testing, configuration management, and security. These discussions are followed by a bibliographic section. The tutorial closes with some final advice for network managers.

Network Management Goals and Functions

An organization's view of network management goals is shaped by two factors:

1. people in the organization depend on the system working,
2. LANs, routers, lines, and other communications resources have costs.

From the organizational vantage point, the ultimate goal of network management is to provide a consistent, predictable, acceptable level of service from the available data communications resources. To achieve this, a network manager must first be able to perform fault detection, isolation, and correction. He must also be able to effect configuration changes with a minimum of disruption, and measure the utilization of system components.

People actually managing networks have a different focus. Network managers are usually evaluated by the availability and performance of their communications systems, even though many factors of net performance are beyond their control. To them, the most important requirement of a network management tool is that it allows the detection and diagnosis of faults before users can call to complain: users (and bosses) can often be placated just by knowing that a network problem has been diagnosed. Another vital network management func- tion is the ability to collect data that justify current or future expenditures for the data communications plant and staff.

Following a section on system monitoring, this tutorial addresses fault, performance, configuration, and security management. By fault management, we mean the detection, diagnosis, and correction of network malfunctions. Under the subject of performance management, we include support for predictable, efficient service, as well as capacity planning and capacity testing. Configuration management includes support for orderly configuration changes (usually, system growth), and local administration of component names and addresses. Security management includes both protecting system components from damage and protecting sensitive information from unintentional or malicious disclosure or corruption.

Readers familiar with the ISO management standards and drafts will note both that we have borrowed heavily from the "OSI Management Framework," except that we have omitted the "account management" function. Account management seems a bit out of place with the other network management functions. The logging required by account management is likely to be done by specialized, dedicated subsystems that are distinct from other network management components. Hence, this tutorial does not cover account management. Rest assured, however, that account management, if required, will be adequately supported and staffed.

For those with a DoD background, security may also seem out of place as a subtopic of network management. Without doubt, communications security is an important issue that should be considered in its own right. Because of the requirements of trust for security mechanisms, security components will probably not be integrated subcomponents of a larger network management system. Nevertheless, because a network manager has a responsibility to protect his system from undue security risks, this tutorial includes a discus- sion on internet security.

System Monitoring

System monitoring is a fundamental aspect of network management. One can divide system monitoring into two rough categories: error detection and baseline monitoring.

System errors, such as misformatted frames or dropped packets, are not in themselves cause for concern. Spikes in error rates, however, should be investigated. It is sound practice to log error rates over time, so that increases can be recognized. Furthermore, logging error rates as a func- tion of traffic rates can be used to detect congestion. Investigate unusual error rates and other anomalies as they are detected, and keep a notebook to record your discoveries.

Day-to-day traffic should be monitored, so that the operational baselines of a system and its components can be determined. As well as being essential for performance management, baseline determination and traffic monitoring are the keys to early fault detection.

A preliminary step to developing baseline measurements is construction of a system map: a graphical representation of the system components and their interfaces. Then, measurements of utilization (i.e., use divided by capacity) are needed. Problems are most likely to arise, and system tuning efforts are most likely to be beneficial, at highly utilized components.

It is worthwhile to develop a source/destination traffic matrix, including a breakdown of traffic between the local system and other internet sites. Both volume and type of traffic should be logged, along with its evolution over time. Of particular interest for systems with diskless workstations is memory swapping and other disk server access. For all systems, broadcast traffic and routing traffic should be monitored. Sudden increases in the variance of delay or the volume of routing traffic may indicate thrashing or other soft failures.

In monitoring a system, long-term averages are of little use. Hourly averages are a better indicator of system use. Variance in utilization and delay should also be tracked. Sudden spikes in variance are tell-tale signs that a problem is looming or exists. So, too, are trends of increased packet or line errors, broadcasts, routing traffic, or delay.

Fault Detection and Isolation

When a system fails, caution is in order. A net manager should make an attempt to diagnose the cause of a system crash before rebooting. In many cases, however, a quick diagnosis will not be possible. For some high priority applications, restoring at least some level of service will have priority over fault repair or even complete fault diagnosis. This necessitates prior planning. A net manager must know the vital applications at his site. If applications require it, he must also have a fall-back plan for bringing them online. Meanwhile, repeated crashes or hardware failures are unambiguous signs of a problem that must be corrected.

A network manager should prepare for fault diagnosis by becoming familiar with how diagnostic tools respond to network failure. In times of relative peace, a net manager should occasionally unplug the network connection from an unused workstation and then "debug" the problem.

When diagnosing a fault or anomaly, it is vital to proceed in an orderly manner, especially since network faults will usually generate spurious as well as accurate error messages. Remember to keep in mind that the network itself is failing. Do not place too much trust in anything obtained remotely. Furthermore, it is unlikely to be significant that remote information such as DNS names or NFS files cannot be obtained. Even spurious messages can be revealing, because they pro- vide clues to the problem. From the data at hand, develop working hypotheses about probable causes of the problems you detect. Direct your further data gathering efforts so that the information you get will either refute or support your hypotheses.

An orderly approach to debugging is facilitated if it is guided by a model of network behavior. The following portions of this section present such a model, along with a procedure for checking network connectivity. The section concludes with some hints for diagnosing a particularly tricky class of connectivity problem.

A Network Model as a Diagnostic Framework

The point of having a model of how things work is to have a basis for developing educated guesses about how things go wrong. The problem of cascading faults -- faults generating other faults -- makes use of a conceptual model a virtual necessity.

In general, only problems in a component's hardware or operating system will generate simultaneous faults in multiple protocol layers. Otherwise, faults will propagate vertically (up the protocol stack) or horizontally (between peer-level communications components). Applying a conceptual model that includes the architectural relations of net- work components can help to order an otherwise senseless barrage of error messages and symptoms.

The model does not have to be formal or complex to bring structure to debugging efforts. A useful start is something as simple as the following:

1. Applications programs use transport services: TCP/UDP. Before using service, applications that accept host names as parameters must translate the names into IP addresses. Translation may be based on a static table lookup (/etc/hosts file in UNIX hosts), the DNS, or yellow pages. Nslookup and DiG are tools for monitoring the activities of the DNS.
2. Transport protocol implementations use IP services. The local IP module makes the initial decision on forwarding. An IP datagram is forwarded directly to the destination host if the destination is on the same network as the source.

   Otherwise, the datagram is forwarded to a gateway attached to the network. On BSD hosts, the contents of a host's routing table are visible by use of the "netstat" command.

3. IP implementations translate the IP address of a datagram's next hop (either the destination host or a gateway) to a local network address. For ethernets, the Address Resolution Protocol (ARP) is commonly used for this translation. On BSD systems, an interface's IP address and other configuration options can be viewed by use of the "ifconfig" command, while the contents of a host's ARP cache may be viewed by use of "arp" command.
4. IP implementations in hosts and gateways route datagrams based on subnet and net identifiers. Subnetting is a means of allocating and preserving IP address space, and of insulating users from the topological details of a multi-network campus. Sites that use subnetting reserve portions of the IP address's host identifier to indicate particular networks at their campus. Subnetting is highly system-dependent. The details are a critical, though local, issue. As for routing between separate networks, a variety of gateway-to-gateway protocols are used. Traceroute is a useful tool for investigating routing problems. The tool, "query," can be used to examine RIP routing tables.

A neophyte network manager should expand the above description so that it accurately describes his particular system, and learn the tools and techniques for monitoring the operations at each of the above stages.

A Simple Procedure for Connectivity Check

In this section, we describe a procedure for isolating a TCP/IP connectivity problem. In this procedure, a series of tests methodically examine connectivity from a host, starting with nearby resources and working outward. The steps in our connectivity-testing procedure are:

1. As an initial sanity check, ping your own IP address and the loopback address.
2. Next, try to ping other IP hosts on the local subnet. Use numeric addresses when starting off, since this eliminates the name resolvers and host tables as potential sources of problems. The lack of an answer may indicate either that the destination host did not respond to ARP (if it is used on your LAN), or that a datagram was forwarded (and hence, the destination IP address was resolved to a local media address) but that no ICMP Echo Reply was received. This could indicate a length-related problem, or misconfigured IP Security.
3. If an IP router (gateway) is in the system, ping both its near and far-side addresses.
4. Make sure that your local host recognizes the gateway as a relay. (For BSD hosts, use netstat.)
5. addresses - Still using numeric IP addresses, try to ping hosts beyond the gateway. If you get no response, run hopcheck or traceroute, if available. Note whether your packets even go to the gateway on their way to the destination,. If not, examine the methods used to instruct your host to use this gateway to reach the specified destination net (e.g., is the default route in place? Alternatively, are you successfully wire-tapping the IGP messages broadcast on the net you are attached to?)

   If traceroute is not available, ping, netstat, arp, and a knowledge of the IP addresses of all the gateway's interfaces can be used to isolate the cause of the problem. Use netstat to determine your next hop to the destination. Ping that IP address to ensure the router is up. Next, ping the router interface on the far subnet. If the router returns "network unreachable" or other errors, investigate the router's routing tables and interface status. If the pings succeed, ping the close interface of the succeeding next hop gateway, and so on. Remember the routing along the outbound and return paths may be different.
6. Once ping is working with numeric addresses, use ping to try to reach a few remote hosts by name. If ping fails when host names are used, check the operation of the local name-mapping system (i.e., with nslookup or DiG). If you want to use "shorthand" forms ("myhost" instead of "myhost.mydomain.com"), be sure that the alias tables are correctly configured.
7. Once basic reachability has been established with ping, try some TCP-based applications: FTP and TELNET are supported on almost all IP hosts, but FINGER is a simpler protocol. The Berkeley-specific protocols (RSH, RCP, REXEC and LPR) require extra configuration on the server host before they can work, and so are poor choices for connectivity testing.

If problems arise in steps 2-7 above, rerunning the tests while executing a line monitor (e.g.,

etherfind, netwatch, or tcpdump) can help to pinpoint the problem.

The above procedure is sound and useful, especially if little is known about the cause of the connectivity problem. It is not, however, guaranteed to be the shortest path to diagnosis. In some cases, a binary search on the problem might be more effective (i.e., try a test "in the middle," in a spot where the failure modes are well defined). In other cases, available information might so strongly suggest a particular failure that immediately testing for it is in order. This last "approach," which might be called "hunting and pecking," should be used with caution: chasing one will o' the wisp after another can waste much time and effort.

Note that line problems are still among the most common causes of connectivity loss. Problems in transmission across local media are outside the scope of this tutorial. But, if a host or workstation loses or cannot establish connectivity, check its physical connection.

Limited Connectivity

An interesting class of problems can result in a particularly mysterious failure: TELNET or other low-volume TCP connections work, but large file transfers fail. FTP transfers may start, but then hang. There are several possible culprits in this problem. The most likely suspects are IP implementations that cannot fragment or reassemble datagrams, and TCP implementations that do not perform dynamic window sizing (a.k.a. Van Jacobson's "Slow Start" algorithm). Another possibility is mixing incompatible frame formats on an ethernet.

Even today, some IP implementations in the Internet cannot correctly handle fragmentation or reassembly. They will work fine for small packets, but drop all large packets.

The problem can also be caused by buffer exhaustion at gateways that connect interfaces of widely differing bandwidth. Datagrams from a TCP connection that traverses a bottleneck will experience queue delays, and will be dropped if buffer resources are depleted. The congestion can be made worse if the TCP implementation at the traffic source does not use the recommended algorithms for computing retransmission times, since spuriously retransmitted datagrams will only add to the congestion. Fragmentation, even if correctly implemented, will compound this problem, since processing delays and congestion will be increased at the bottleneck.

Serial Line Internet Protocol (SLIP) links are especially vulnerable to this and other congestion problems. SLIP lines are typically an order of magnitude slower than other gateway interfaces. Also, the SLIP lines are at times configured with MTUs (Maximum Transfer Unit, the maximum length of an IP datagram for a particular subnet) as small as 256 bytes, which virtually guarantees fragmentation.

To alleviate this problem, TCP implementations behind slow lines should advertise small windows. Also, if possible, SLIP lines should be configured with an MTU no less than 576 bytes. The tradeoff to weigh is whether interactive traffic will be penalized too severly by transmission delays of lengthy datagrams from concurrent file transfers.

Misuse of ethernet trailers can also cause the problem of hanging file transfers. "Trailers" refers to an ethernet frame format optionally employed by BSD systems to minimize buffer copying by system software. BSD systems with ethernet interfaces can be configured to send large frames so that their address and control data are at the end of a frame (hence, a "trailer" instead of a "header"). After a memory page is allocated and loaded with a received ethernet frame, the ethernet data will begin at the start of the memory page boundary. Hence, the ethernet control informa- tion can be logically stripped from the end merely by adjusting the page's length field. By manipulating virtual memory mapping, this same page (sans ethernet control infor- mation), can then be passed to the local IP module without additional allocation and loading of memory. The disadvan- tage in using trailers is that it is non-standard. Many implementations cannot parse trailers.

The hanging FTP problem will appear if a gateway is not configured to recognize trailers, but a host or gateway immediately "upstream" on an ethernet uses them. Short datagrams will not be formatted with trailers, and so will be processed correctly. When the bulk data transfer starts, how- ever, full-sized frames will be sent, and will use the trailer format. To the gateway that receives them, they appear simply as misformatted frames, and are quietly dropped. The solution, obviously, is to insure that all hosts and gateways on an ethernet are consistent in their use of trailers. Note that RFC 1122, "Internet Host Requirements," places very strict restrictions on the use of trailers.

Performance Testing

Performance management encompasses two rather different activities. One is passive system monitoring to detect problems and determine operational baselines. The goal is to measure system and component utilization and so locate bottlenecks, since bottlenecks should receive the focus of performance tuning efforts. Also, performance data is usually required by upper level management to justify the costs of communications systems. This is essentially identical to system monitoring, and is addressed at greater length in Section 2, above.

Another aspect of performance management is active performance testing and capacity planning. Some work in this area can be based on analysis. For example, a rough estimate of gateway capacity can be deduced from a simple model given by Charles Hedrick in his "Introduction to Administration of an Internet-based Local Network," which is

per-packet processing time = switching time + (packet size) * (transmission bps).

Another guideline for capacity planning is that in order to avoid excessive queuing delays, a system should be sized at about double its expected load. In other words, system capacity should be so high that utilization is no greater than 50%.

Although there are more sophisticated analytic models of communications systems than those above, their added complexity does not usually gain a corresponding accuracy. Most analytic models of communications nets require assumptions about traffic load distributions and service rates that are not merely problematic, but are patently false. These errors tend to result in underestimating queuing delays. Hence, it is often necessary to actually load and measure the performance of a real communications system if one is to get accurate performance predictions. Obviously, this type of testing is performed on isolated systems or during off hours. The results can be used to evaluate parameter settings or predict performance during normal operations.

Simulations can be used to supplement the testing of real systems. To be believable, however, simulations require validation, which, in turn, requires measurements from a real system. Whether testing or simulating a system's performance, actual traffic traces should be incorporated as input to traffic generators. The performance of a communi- cations system will be greatly influenced by its load characteristics (burstiness, volume, etc.), which are them- selves highly dependent on the applications that are run.

When tuning a net, in addition to the usual configuration parameters, consider the impact of the location of gateways and print and file servers. A few rules of thumb can guide the location of shared system resources. First, there is the principle of locality: a system will perform better if most traffic is between nearby destinations. The second rule is to avoid creating bottlenecks. For example, multiple diskservers may be called for to support a large number of workstations. Furthermore, to avoid LAN and diskserver congestion, workstations should be configured with enough memory to avoid frequent swapping.

As a final note on performance management, proceed cautiously if your ethernet interface allows you to customize its collision recovery algorithm. This is almost always a bad idea. The best that it can accomplish is to give a few favored hosts a disproportionate share of the ethernet bandwidth, perhaps at the cost of a reduction in total sys- tem throughput. Worse, it is possible that differing collision recovery algorithms may exhibit a self-synchronizing behavior, so that excess collisions are generated.

Configuration Management

Configuration management is the setting, collecting, and storing of the state and parameters of network resources. It overlaps all other network management functions. Hence, some aspects of configuration management have already been addressed (e.g., tuning for performance). In this section, we will focus on configuration management activities needed to "hook up" a net or campus to a larger internet. We will not, of course, include specific details on installing or maintaining internetted communications systems. We will, however, skim over some of the TCP/IP configuration highlights.

Configuration management includes "name management" -- the control and allocation of system names and addresses, and the translation between names and addresses. Name-toaddress translation is performed by "name servers." We con- clude this section with a few strictures on the simultaneous use of two automated name-servers, the Domain Name System (DNS), and Yellow Pages (YP).

Required Host Configuration Data for TCP/IP internets

In a TCP/IP internet, each host needs several items of information for internet communications. Some will be host-specific, while other information will be common for all hosts on a subnet. In a soon to be published RFC document, R. Droms identifies the following configuration data required by internet hosts:

- An IP address, a host specific value that can be hard-coded or obtained via BOOTP, the Reverse Address Resolution Protocol (RARP) or Dynamic RARP (DRARP).
- Subnet properties, such as the subnet mask and the Maximum Transmission Unit (MTU); obviously, these values are not host-specific.
- Addresses of "entry" gateways to the internet; addresses of default gateways are usually hard coded; though the ICMP "redirect" message can be used to refine a host's routing tables, there is currently no dynamic TCP/IP mechanism or protocol for a host to locate a gateway; an IETF working group is busy on this problem.
- For hosts in internets using the Domain Name System (DNS) for name-to-address translation, the location of a local DNS server is needed; this information is not host-specific, and usually hard-coded;
- Host name (domain name, for hosts using DNS); obviously host-specific; either hard-coded or obtained in a boot procedure.
- For diskless hosts, various boot services. BOOTP is the standard Internet protocol for downloading boot configuration information. The Trivial File Transfer Protocol (TFTP) is typically used for downloading boot images. Sun computers use the "bootparams" RPC mechanism for downloading initial configuration data to a host.

There are ongoing developments, most notably the work of the Dynamic Host Configuration Working Group of the IETF, to support dynamic, automatic gathering of the above data. In the meantime, most systems will rely on hand-crafted configuration files.

Connecting to THE Internet

The original TCP/IP Internet (spelled with an upper-case "I") is still active, and still growing. An interesting aspect of the Internet is that it spans many independently administered systems.

Connection to the Internet requires: a registered network number, for use in IP addresses; a registered autonomous system number (ASN), for use in internet routing; and, a registered domain name. Fielding a primary and backup DNS server is a condition for registering a domain name.

The Defense Data Network (DDN) Network Information Center (NIC) is responsible for registering network numbers, autonomous system numbers, and domain names. Regional nets will have their own policies and requirements for Internet connections, but all use the NIC for this registration service. Contact the NIC for further information, at:

DDN Network Information Center
SRI International, Room EJ291
333 Ravenswood Avenue
Menlo Park, CA 94025

Email: HOSTMASTER@NIC.DDN.MIL
Phone: 1-415-859-3695 1-800-235-3155 (toll-free hotline)

YP and DNS: Dueling name servers.

The Domain Name System (DNS) provides name service: it translates host names into IP addresses (this mapping is also called "resolution"). Two widespread DNS implementations are "bind" and "named." The Sun Yellow Pages (YP) system can be configured to provide an identical service, by providing remote, keyed access to the "hosts.byname" map. Unfortunately, if both DNS and the YP hosts.byname map are installed, they can interact in disruptive ways.

The problem has been noted in systems in which DNS is used as a fallback, to resolve hostnames that YP cannot. If DNS is slow in responding, the timeout in program ypserv may expire, which triggers a repeated request. This can result in disaster if DNS was initially slow because of congestion: the slower things get, the more requests are generated, which slows things even more. A symptom of this problem is that failures by the DNS server or network will trigger numerous requests to DNS.

Reportedly, the bug in YP that results in the avalanche of DNS requests has been repaired in SunOS 4.1. The problem, however, is more fundamental than an implementation error. The YP map hosts.byname and the DNS contain the same class of information. One can get an answer to the same query from each system. These answers may well be different: there is not a mechanism to maintain consistency between the systems. More critical, however, is the lack of a mechanism or procedure to establish which system is authoritative. Hence, running the DNS and YP name services in parallel is pointless. If the systems stay consistent, then only one is needed. If they differ, there is no way to choose which is correct.

The YP hosts.byname service and DNS are comparable, but incompatible. If possible, a site should not run both services. Because of Internet policy, sites with Internet con- nections MUST use the DNS. If YP is also used, then it should be configured with YP-to-DNS disabled.

Hacking a system so that it uses DNS rather than the YP hosts.byname map is not trivial, and should not be attempted by novices. The approach is to rebuild the shared C linklibrary, so that system calls to gethostbyname() and gethostbyaddr() will use DNS rather than YP. To complete the change, programs that do not dynamically link the shared C library (rcp, arp, etc.) must also be rebuilt.

Modified shared C libraries for Sun 3s and Sun 4s are available via anonymous FTP from host uunet.uu.net, in the sun- fixes directory. Note that use of DNS routines rather than YP for general name resolution is not a supported SunOS feature at this time.

Internet Security

The guidelines and advice in this section pertain to enhancing the protection of data that are merely "sensitive." By themselves, these measures are insufficient for protecting "classified" data. Implementing the policies required to protect classified data is subject to stringent, formal review procedures, and is regulated by agencies such as the Defense Investigative Service (DIS) and the National Secu- rity Agency (NSA).

A network manager must realize that he is responsible for protecting his system and its users. Furthermore, though the Internet may appear to be a grand example of a cooperative joint enterprise, recent incidents have made it clear that not all Internet denizens are benign.

A network manager should be aware that the network services he runs have a large impact on the security risks to which his system is exposed. The prudent network manager will be very careful as to what services his site provides to the rest of the Internet, and what access restrictions are enforced. For example, the protocol "finger" may provide more information about a user than should be given to the world at large. Worse, most implementations of the protocol TFTP give access to all world-readable files.

This section highlights several basic security considerations for Internet sites. It then lists several sources of information and advice on improving the security of systems connected to the Internet.

Basic Internet Security

Two major Internet security threats are denial of service and unauthorized access.

Denial of service threats often take the form of protocol spoofers or other malicious traffic generators. These problems can be detected through system monitoring logs. If an attack is suspected, immediately contact your regional net office (e.g., SURANET, MILNET). In addition, DDN users should contact SCC, while other Internet users should con- tact CERT (see below). A cogent description of your system's symptoms will be needed.

At your own site, be prepared to isolate the problems (e.g., by limiting disk space available to the message queue of a mail system under attack). As a last resort, coping with an attack may require taking down an Internet connection. It is better, however, not to be too quick to quarantine your site, since information for coping with the attack may come via the Internet.

Unauthorized access is a potentially more ominous security threat. The main avenues are attacks against passwords and attacks against privileged system processes.

An appallingly common means of gaining entry to systems is by use of the initial passwords to root, sysdiag, and other management accounts that systems are shipped with. Only slightly less vulnerable are common or trivial passwords, since these are readily subverted by dictionary attacks. Obvious steps can reduce the risk of password attacks: passwords should be short-lived, at least eight characters long, with a mix of upper and lower case, and preferably random. The distasteful aspect of memorizing a random string can be alleviated if the password is pronounceable.

Improving passwords does not remove all risks. Passwords transmitted over an ethernet are visible to all attached systems. Furthermore, gateways have the potential to intercept passwords used by any FTP or TELNET connections that traverse them. It is a bad idea for the root account to be accessed by FTP or TELNET if the connections must cross untrusted elements.

Attacks against system processes are another avenue of unauthorized access. The principle is that by subverting a sys- tem process, the attacker can then gain its access privileges.

One approach to reducing this risk is to make system programs harder to subvert. For example, the widespread attack in November 1988 by a self-replicating computer program ("worm," analogous to a tapeworm) subverted the "fingerd" process, by loading an intrusive bootstrap program (known variously as a "grappling hook" or "vector" program), and then corrupting the stack space so that a subroutine's return address was overwritten with the address of the bootstrap program. The security hole in fingerd consisted of an input routine that did not have a length check. Security fixes to "fingerd" include the use of a revised input routine.

A more general protection is to apply the principle of "least privilege." Where possible, system routines should run under separate user IDs, and should have no more privilege than is necessary for them to function.

To further protect against attacks on system processes, system managers should regularly check their system programs to ensure that they have not been tampered with or modified in any way. Checksums should be used for this purpose. Using the operating system to check a file's last date of modifi- cation is insufficient, since the date itself can be compromised.

Finally, to avoid the unauthorized replacement of system code, care should be exercised in

assigning protection to its directory paths.

Some system programs actually have "trap doors" that facilitate subversion. A trap door is the epitome of an undocu- mented feature: it is a hidden capability of a system pro- gram that allows a knowledgeable person to gain access to a system. The Internet Worm exploited what was essentially a trap door in the BSD sendmail program.

Ensuring against trap doors in software as complex as sendmail may be infeasible. In an ideal world, the BSD sendmail program would be replaced by an entire mail subsystem (i.e., perhaps including mail user agents, mail transfer agents, and text preparation and filing programs). Any site using sendmail should at least obtain the less vulnerable, toughened distribution from ucbarpa.berkeley.edu, in file ~ftp/4.3/sendmail.tar.Z. Sites running SunOS should note that the 4.0.3 release closed the security holes exploited by the Internet Worm. Fixes for a more obscure security hole in SunOS are available from host uunet.uu.net in ~ftp/sun-fixes; these improvements have been incorporated in SunOS 4.1.

Sendmail has problems other than size and complexity. Its use of root privileges, its approach to alias expansion, and several other design characteristics present potential avenues of attack. For UNIX sites, an alternative mail server to consider is MMDF, which is now at version 2. MMDF is distributed as part of the SCO UNIX distribution, and is also available in the user contributed portion of 4.3BSD. Though free, MMDF is licensed, and resale is restricted. Sites running MMDF should be on the mmdf email list; requests to join this list should be sent to: mmdf2-request@relay.cs.net.

Programs that masquerade as legitimate system code but which contain trap doors or other aides to unauthorized access are known as trojan horses. Computer "viruses," intrusive software that infects seemingly innocent programs and propagates when the infected programs are executed or copied, are a special case of trojan horse programs.*

To guard against trojan horse attacks, be wary of programs downloaded from remote sources. At minimum, do not download executables from any but the most trusted sources. Also, as noted above, to avoid proliferation of "infected" software, checksums should be computed, recorded, and periodically verified.

Security Information Clearing-Houses

The Internet community can get security assistance from the Computer Emergency Response Team (CERT), established by DARPA in November 1988. The Coordination Center for the CERT (CERT/CC) is located at the Software Engineering Institute at Carnegie Mellon University. The CERT is intended to respond to computer security threats such as the November '88 worm attack that invaded many defense and research com- puters. Consult RFC 1135 (Reynolds, J., "The Helminthiasis of the Internet", USC/ISI, December 1989), for further information.

CERT assists Internet sites in response to security attacks or other emergency situations. It can immediately tap experts to diagnose and solve the problems, as well as establish and maintain communications with the affected computer users and with government authorities as appropriate. Specific responses will be taken in accordance with the nature of the problem and the magnitude of the threat.

CERT is also an information clearing-house for the identification and repair of security vulnerabilities, informal assessments of existing systems in the research community, improvement to emergency response capability, and both vendor and user security awareness. This security information is distributed by periodic bulletins, and is posted to the USENET news group comp.security.announce. In addition, the security advisories issued by CERT, as well as other useful security-related information, are available via anonymous FTP from cert.sei.cmu.edu.

For immediate response to attacks or incidents, CERT mans a 24-hour hotline at (412) 268-7090. To subscribe to CERT's security announcement bulletin, or for further information, contact:

> CERT
> Software Engineering Institute
> Carnegie Mellon University
> Pittsburgh, PA 15213-3890
> (412) 268-7080
> cert@cert.sei.cmu.edu.

For DDN users, the Security Coordination Center (SCC) serves a function similar to CERT. The SCC is the DDN's clearinghouse for host/user security problems and fixes, and works with the DDN Network Security Officer. The SCC also distributes the DDN Security Bulletin, which communicates information on network and host security exposures, fixes, and con- cerns to security and management personnel at DDN facili- ties. It is available online, via kermit or anonymous FTP, from nic.ddn.mil, in SCC:DDN-SECURITY-yy-nn.TXT (where "yy" is the year and "nn" is the bulletin number). The SCC provides immediate assistance with DDN-related host security problems; call (800) 235-3155 (6:00 a.m. to 5:00 p.m. Pacific Time) or send e-Mail to SCC@NIC.DDN.MIL. For 24 hour coverage, call the MILNET Trouble Desk (800) 451-7413 or AUTOVON 231-1713.

The CERT/CC and the SCC communicate on a regular basis and support each other when problems occur. These two organizations are examples of the incident response centers that are forming; each serving their own constituency or focusing on a particular area of technology.

Other network groups that discuss security issues are: comp.protocols.tcp-ip, comp.virus (mostly PC-related, but occasionally covers Internet topics), misc.security, and the BITNET Listserv list called VIRUS-L.

Internet Information

There are many available references on the TCP/IP protocol suite, the internet architecture, and the DDN Internet. A soon to be published FYI RFC document, "Where to Start: A Bibliography of General Internetworking Information." provides a bibliography of online and hard copy documents, reference materials, and multimedia training tools that address general networking information and "how to use the Internet." It presents a representative collection of materials that will help the reader become familiar with the concepts of internetworking. Inquires on the current status of this document can be sent to user-doc@nnsc.nsf.net or by postal mail to:

> Corporation for National Research Initiatives
> 1895 Preston White, Suite 100
> Reston, VA 22091
> Attn: IAB Secretariat.

Two texts on networking are especially noteworthy. Internetworking With TCP/IP, by Douglas Comer, is an informative description of the TCP/IP protocol suite and its underlying architecture. The UNIX System Administration Handbook, by Nemeth, Snyder, and Seebass, is a "must have" for system administrators who are responsible for UNIX hosts. In addi- tion to covering UNIX, it provides a wealth of tutorial material on networking, the Internet, and network management.

A great deal of information on the Internet is available online. An automated, online reference service is available from CSNET. To obtain a bibliography of their online offerings, send the email message

> request: info
> topic: help
> request: end

to info-server@sh.cs.net.

The DDN NIC also offers automated access to many NIC documents, online files, and WHOIS information via electronic mail. To use the service, send an email message with your request specified in the SUBJECT field of the message. For a sampling of the type of offerings available through this service, send the following message

> To: SERVICE@NIC.DDN.MIL
> Subject: help
> Msg: <none>

The DDN Protocol Implementations and Vendors Guide, published by the DDN Network Information Center (DDN NIC), is an online reference to products and implementations associated with the DoD Defense Data Network (DDN) group of com- munication protocols, with emphasis on TCP/IP and OSI proto- cols. It contains information on protocol policy and evaluation procedures, a discussion of software and hardware implementations, and analysis tools with a focus on protocol and network analyzers. To obtain the guide, invoke FTP at your local host and connect to host NIC.DDN.MIL (internet address 26.0.0.73 or 10.0.0.51). Log in using username 'anonymous' with password 'guest' and get the file NETINFO:VENDORS-GUIDE.DOC.

The DDN Protocol Guide is also available in hardcopy form. To obtain a hardcopy version of the guide, contact the DDN Network Information Center:

> By U.S. mail:

SRI International
DDN Network Information Center
333 Ravenswood Avenue, Room EJ291
Menlo Park, CA 94025

By e-mail:

NIC@NIC.DDN.MIL

By phone:

1-415-859-3695
1-800-235-3155 (toll-free hotline)

For further information about the guide, or for information on how to list a product in a subsequent edition of the guide, contact the DDN NIC.

There are many additional online sources on Internet Management. RFC 1118, "A Hitchhiker's Guide to the Internet," by Ed Krol, is a useful introduction to the Internet routing algorithms. For more of the nitty-gritty on laying out and configuring a campus net, see Charles Hedrick's "Introduction to Administration of an Internet-based Local Network," available via anonymous FTP from cs.rutgers.edu (sometimes listed in host tables as aramis.rutgers.edu), in subdirectory runet, file tcp-ip-admin. Finally, anyone responsible for systems connected to the Internet must be thoroughly versed in the Host Requirements RFCs (RFC 1122 and RFC 1123) and "Requirements for Internet Gateways," RFC 1009.

The Final Words on Internet Management

Keep smiling, no matter how bad things may seem. You are the expert. They need you.

Author's Address

Robert H. Stine
SPARTA, Inc.
7926 Jones Branch Drive
Suite 1070
McLean, VA 22102

EMail: STINE@SPARTA.COM

Initial forwarding may actually be complex and vulnerable to multiple points of failure. For example, when sending an IP datagram, 4.3BSD hosts first look for a route to the particular host. If none has been specified for the destination, then a search is made for a route to the network of the destination. If this search also fails, then as a last resort, a search is made for a route to a "default" gateway. Routes to hosts, networks, and the "default" gateway may be static, loaded at boot time and perhaps updated by operator commands. Alternatively, they may be dynamic, loaded from redirects and routing protocol updates.

Thanks to James VanBokkelen, president of FTP Software, for sharing with us a portion of a PC/TCP support document, the basis for the above connectivity procedure.

To avoid this problem, TCP implementations on the In- ternet must use "exponential backoff" between succes- sive retransmissions, Karn's algorithm for filtering samples used to estimate round-trip delay between TCP peers, and Jacobson's algorithm for incorporating vari- ance into the "retransmission time-out" computation for TCP segments. See Section 4.2.3.1 of RFC 1122, "Requirements for Internet Hosts Communication Layers."

Draft "Dynamic Configuration of Internet Hosts."

Exotic fantasy creatures and women's names are well represented in most password dictionaries.

An early account of the Internet Worm incident of November 1988 is given by Eugene Spafford in the January 89 issue of "Computer Communications Review." Several other articles on the worm incident are in the June 89 issue of the "Communications of the ACM."

Virus attacks have been seen against PCs, but as yet have rarely been directed agains UNIX systems.

Products mentioned in the guide are not specifically endorsed or recommended by the Defense Communications Agency (DCA).

RFC-1155 Structure and Identification of Management Information for TCP/IP Based Internets

Marshall Rose & Keith McCloghrie

May 1990

Status of this Memo

This RFC is a re-release of RFC 1065, with a changed "Status of this Memo", plus a few minor typographical corrections. The technical content of the document is unchanged from RFC 1065.

This memo provides the common definitions for the structure and identification of management information for TCP/IP-based internets. In particular, together with its companion memos which describe the management information base along with the network management protocol, these documents provide a simple, workable architecture and system for managing TCP/IP-based internets and in particular, the Internet.

This memo specifies a Standard Protocol for the Internet community. Its status is "Recommended". TCP/IP implementations in the Internet which are network manageable are expected to adopt and implement this specification.

The Internet Activities Board recommends that all IP and TCP implementations be network manageable. This implies implementation of the Internet MIB (RFC-1156) and at least one of the two recommended management protocols SNMP (RFC-1157) or CMOT (RFC-1189). It should be noted that, at this time, SNMP is a full Internet standard and CMOT is a draft standard. See also the Host and Gateway Requirements RFCs for more specific information on the applicability of this standard.

Please refer to the latest edition of the "IAB Official Protocol Standards" RFC for current information on the state and status of standard Internet protocols.

Distribution of this memo is unlimited.

RFC-1155 Structure and Identification of Management Information

Introduction

This memo describes the common structures and identification scheme for the definition of management information used in managing TCP/IP-based internets. Included are descriptions of an object information model for network management along with a set of generic types used to describe management information. Formal descriptions of the structure are given using Abstract Syntax Notation One (ASN.1) [1].

This memo is largely concerned with organizational concerns and administrative policy: it neither specifies the objects which are managed, nor the protocols used to manage those objects. These concerns are addressed by two companion memos: one describing the Management Information Base (MIB) [RFC-1156], and the other describing the Simple Network Management Protocol (SNMP) [RFC-1157].

This memo is based in part on the work of the Internet Engineering Task Force, particularly the working note titled "Structure and Identification of Management Information for the Internet" [4]. This memo uses a skeletal structure derived from that note, but differs in one very significant way: that note focuses entirely on the use of OSI-style network management. As such, it is not suitable for use with SNMP.

This memo attempts to achieve two goals: simplicity and extensibility. Both are motivated by a common concern: although the management of TCP/IP-based internets has been a topic of study for some time, the authors do not feel that the depth and breadth of such understanding is complete. More bluntly, we feel that previous experiences, while giving the community insight, are hardly conclusive. By fostering a simple SMI, the minimal number of constraints are imposed on future potential approaches; further, by fostering an extensible SMI, the maximal number of potential approaches are available for experimentation.

It is believed that this memo and its two companions comply with the guidelines set forth in RFC 1052, "IAB Recommendations for the Development of Internet Network Management Standards" [5] and RFC 1109, "Report of the Second Ad Hoc Network Management Review Group" [6]. In particular, we feel that this memo, along with the memo describing the management information base, provide a solid basis for network management of the Internet.

Structure and Identification of Management Information

Managed objects are accessed via a virtual information store, termed the Management Information Base or MIB. Objects in the MIB are defined using Abstract Syntax Notation One (ASN.1) [1].

Each type of object (termed an object type) has a name, a syntax, and an encoding. The name is represented uniquely as an OBJECT IDENTIFIER. An OBJECT IDENTIFIER is an administratively assigned name. The administrative policies used for assigning names are discussed later in this memo.

The syntax for an object type defines the abstract data structure corresponding to that object type. For example, the structure of a given object type might be an INTEGER or OCTET STRING. Although in general, we should permit any ASN.1 construct to be available for use in defining the syntax of an object type, this memo purposely restricts the ASN.1 constructs which may be used. These restrictions are made solely for the sake of simplicity.

The encoding of an object type is simply how instances of that object type are represented using the object's type syntax. Implicitly tied to the notion of an object's syntax and encoding is how the object is represented when being transmitted on the network. This memo specifies the use of the basic encoding rules of ASN.1 [7].

It is beyond the scope of this memo to define either the MIB used for network management or the network management protocol. As mentioned earlier, these tasks are left to companion memos. This memo attempts to minimize the restrictions placed upon its companions so as to maximize generality. However, in some cases, restrictions have been made (e.g., the syntax which may be used when defining object types in the MIB) in order to encourage a particular style of management. Future editions of this memo may remove these restrictions.

RFC-1155 Structure and Identification of Management Information - Structure

Names

Names are used to identify managed objects. This memo specifies names which are hierarchical in nature. The OBJECT IDENTIFIER concept is used to model this notion. An OBJECT IDENTIFIER can be used for purposes other than naming managed object types; for example, each international standard has an OBJECT IDENTIFIER assigned to it for the purposes of identification. In short, OBJECT IDENTIFIERs are a means for identifying some object, regardless of the semantics associated with the object (e.g., a network object, a standards document, etc.)

An OBJECT IDENTIFIER is a sequence of integers which traverse a global tree. The tree consists of a root connected to a number of labeled nodes via edges. Each node may, in turn, have children of its own which are labeled. In this case, we may term the node a subtree. This process may continue to an arbitrary level of depth. Central to the notion of the OBJECT IDENTIFIER is the understanding that administrative control of the meanings assigned to the nodes may be delegated as one traverses the tree. A label is a pairing of a brief textual description and an integer.

The root node itself is unlabeled, but has at least three children directly under it: one node is administered by the International Organization for Standardization, with label iso(1); another is administrated by the International Telegraph and Telephone Consultative Committee, with label ccitt(0); and the third is jointly administered by the ISO and the CCITT, joint-iso-ccitt(2).

Under the iso(1) node, the ISO has designated one subtree for use by other (inter)national organizations, org(3). Of the children nodes present, two have been assigned to the U.S. National Institutes of Standards and Technology. One of these subtrees has been transferred by the NIST to the U.S. Department of Defense, dod(6).

As of this writing, the DoD has not indicated how it will manage its subtree of OBJECT IDENTIFIERs. This memo assumes that DoD will allocate a node to the Internet community, to be administered by the Internet Activities Board (IAB) as follows:

```
internet      OBJECT IDENTIFIER ::= { iso org(3) dod(6) 1 }
```

That is, the Internet subtree of OBJECT IDENTIFIERs starts with the prefix:

```
1.3.6.1.
```

This memo, as a standard approved by the IAB, now specifies the policy under which this subtree of OBJECT IDENTIFIERs is administered. Initially, four nodes are present:

```
directory     OBJECT IDENTIFIER ::= { internet 1 }
mgmt          OBJECT IDENTIFIER ::= { internet 2 }
experimental  OBJECT IDENTIFIER ::= { internet 3 }
private       OBJECT IDENTIFIER ::= { internet 4 }
```

Directory
Mgmt
Experimental
Private

Directory

The directory(1) subtree is reserved for use with a future memo that discusses how the OSI Directory may be used in the Internet.

RFC-1155 Structure and Identification of Management Information - Names

Mgmt

The mgmt(2) subtree is used to identify objects which are defined in IAB-approved documents. Administration of the mgmt(2) subtree is delegated by the IAB to the Internet Assigned Numbers Authority for the Internet. As RFCs which define new versions of the Internet- standard Management Information Base are approved, they are assigned an OBJECT IDENTIFIER by the Internet Assigned Numbers Authority for identifying the objects defined by that memo.

For example, the RFC which defines the initial Internet standard MIB would be assigned management document number 1. This RFC would use the OBJECT IDENTIFIER

    { mgmt 1 }

or

    1.3.6.1.2.1

in defining the Internet-standard MIB.

The generation of new versions of the Internet-standard MIB is a rigorous process. Section 5 of this memo describes the rules used when a new version is defined.

Experimental

The experimental(3) subtree is used to identify objects used in Internet experiments. Administration of the experimental(3) subtree is delegated by the IAB to the Internet Assigned Numbers Authority of the Internet.

For example, an experimenter might received number 17, and would have available the OBJECT IDENTIFIER

    { experimental 17 }

or

    1.3.6.1.3.17

for use.

As a part of the assignment process, the Internet Assigned Numbers Authority may make requirements as to how that subtree is used.

RFC-1155 Structure and Identification of Management Information - Names

Private

The private(4) subtree is used to identify objects defined unilaterally. Administration of the private(4) subtree is delegated by the IAB to the Internet Assigned Numbers Authority for the Internet. Initially, this subtree has at least one child:

enterprises OBJECT IDENTIFIER ::= { private 1 }

The enterprises(1) subtree is used, among other thigs, to permit parties providing networking subsystems to register models of their products.

Upon receiving a subtree, the enterprise may, for example, define new MIB objects in this subtree. In addition, it is strongly recommended that the enterprise will also register its networking subsystems under this subtree, in order to provide an unambiguous identification mechanism for use in management protocols. For example, if the "Flintstones, Inc." enterprise produced networking subsystems, then they could request a node under the enterprises subtree from the Internet Assigned Numbers Authority. Such a node might be numbered:

1.3.6.1.4.1.42

The "Flintstones, Inc." enterprise might then register their "Fred Router" under the name of:

1.3.6.1.4.1.42.1.1

Syntax

Syntax is used to define the structure corresponding to object types. ASN.1 constructs are used to define this structure, although the full generality of ASN.1 is not permitted.

The ASN.1 type ObjectSyntax defines the different syntaxes which may be used in defining an object type.

Primitive Types

Only the ASN.1 primitive types INTEGER, OCTET STRING, OBJECT IDENTIFIER, and NULL are permitted. These are sometimes referred to as non-aggregate types.

RFC-1155 Structure and Identification of Management Information - Syntax

Guidelines for Enumerated INTEGERs

If an enumerated INTEGER is listed as an object type, then a named- number having the value 0 shall not be present in the list of enumerations. Use of this value is prohibited.

Constructor Types

The ASN.1 constructor type SEQUENCE is permitted, providing that it is used to generate either lists or tables.

For lists, the syntax takes the form:

SEQUENCE { <type1>, ..., <typeN> }

where each <type> resolves to one of the ASN.1 primitive types listed above. Further, these ASN.1 types are always present (the DEFAULT and OPTIONAL clauses do not appear in the SEQUENCE definition).

For tables, the syntax takes the form:

SEQUENCE OF <entry>

where <entry> resolves to a list constructor.

Lists and tables are sometimes referred to as aggregate types.

Defined Types

In addition, new application-wide types may be defined, so long as they resolve into an IMPLICITly defined ASN.1 primitive type, list, table, or some other application-wide type. Initially, few application-wide types are defined. Future memos will no doubt define others once a consensus is reached.

NetworkAddress

This CHOICE represents an address from one of possibly several protocol families.
Currently, only one protocol family, the Internet family, is present in this CHOICE.

RFC-1155 Structure and Identification of Management Information - Defined Types

IpAddress

This application-wide type represents a 32-bit internet address. It is represented as an OCTET STRING of length 4, in network byte-order.

When this ASN.1 type is encoded using the ASN.1 basic encoding rules, only the primitive encoding form shall be used.

RFC-1155 Structure and Identification of Management Information - Defined Types

Counter

This application-wide type represents a non-negative integer which monotonically increases until it reaches a maximum value, when it wraps around and starts increasing again from zero. This memo specifies a maximum value of 2^32-1 (4294967295 decimal) for counters.

Gauge

This application-wide type represents a non-negative integer, which may increase or decrease, but which latches at a maximum value. This memo specifies a maximum value of 2^32-1 (4294967295 decimal) for gauges.

RFC-1155 Structure and Identification of Management Information - Defined Types

TimeTicks

This application-wide type represents a non-negative integer which counts the time in hundredths of a second since some epoch. When object types are defined in the MIB which use this ASN.1 type, the description of the object type identifies the reference epoch.

Opaque

This application-wide type supports the capability to pass arbitrary ASN.1 syntax. A value is encoded using the ASN.1 basic rules into a string of octets. This, in turn, is encoded as an OCTET STRING, in effect "double-wrapping" the original ASN.1 value.

Note that a conforming implementation need only be able to accept and recognize opaquely-encoded data. It need not be able to unwrap the data and then interpret its contents.

Further note that by use of the ASN.1 EXTERNAL type, encodings other than ASN.1 may be used in opaquely-encoded data.

Encodings

Once an instance of an object type has been identified, its value may be transmitted by applying the basic encoding rules of ASN.1 to the syntax for the object type.

Managed Objects

Although it is not the purpose of this memo to define objects in the MIB, this memo specifies a format to be used by other memos which define these objects.

An object type definition consists of five fields:

OBJECT:

A textual name, termed the OBJECT DESCRIPTOR, for the object type, along with its corresponding OBJECT IDENTIFIER.

Syntax:

The abstract syntax for the object type. This must resolve to an instance of the ASN.1 type ObjectSyntax (defined below).

Definition:

A textual description of the semantics of the object type. Implementations should ensure that their instance of the object fulfills this definition since this MIB is intended for use in multi-vendor environments. As such it is vital that objects have consistent meaning across all machines.

Access:

One of read-only, read-write, write-only, or not-accessible.

Status:

One of mandatory, optional, or obsolete.

Future memos may also specify other fields for the objects which they define.

RFC-1155 Structure and Identification of Management Information - Managed Objects

Guidelines for Object Names

No object type in the Internet-Standard MIB shall use a sub- identifier of 0 in its name. This value is reserved for use with future extensions.

Each OBJECT DESCRIPTOR corresponding to an object type in the internet-standard MIB shall be a unique, but mnemonic, printable string. This promotes a common language for humans to use when discussing the MIB and also facilitates simple table mappings for user interfaces.

RFC-1155 Structure and Identification of Management Information - Managed Objects

Object Types and Instances

An object type is a definition of a kind of managed object; it is declarative in nature. In contrast, an object instance is an instantiation of an object type which has been bound to a value. For example, the notion of an entry in a routing table might be defined in the MIB. Such a notion corresponds to an object type; individual entries in a particular routing table which exist at some time are object instances of that object type.

A collection of object types is defined in the MIB. Each such subject type is uniquely named by its OBJECT IDENTIFIER and also has a textual name, which is its OBJECT DESCRIPTOR. The means whereby object instances are referenced is not defined in the MIB. Reference to object instances is achieved by a protocol-specific mechanism: it is the responsibility of each management protocol adhering to the SMI to define this mechanism.

An object type may be defined in the MIB such that an instance of that object type represents an aggregation of information also represented by instances of some number of "subordinate" object types. For example, suppose the following object types are defined in the MIB:

OBJECT:

atIndex { atEntry 1 }

Syntax:

INTEGER

Definition:

The interface number for the physical address.

Access:

read-write.

Status:

mandatory.

OBJECT:

atPhysAddress { atEntry 2 }

Syntax:

OCTET STRING

Definition:

The media-dependent physical address.

Access:

read-write.

Status:

mandatory.

OBJECT:

atNetAddress { atEntry 3 }

Syntax:

NetworkAddress

Definition:

The network address corresponding to the media-dependent physical address.

Access:

read-write.

Status:

mandatory.

Then, a fourth object type might also be defined in the MIB:

OBJECT:

atEntry { atTable 1 }

Syntax:

```
AtEntry ::= SEQUENCE {
    atIndex
    INTEGER,
    atPhysAddress
    OCTET STRING,
    atNetAddress
    NetworkAddress
    }
```

Definition:

An entry in the address translation table.

Access:

read-write.

Status:

mandatory.

Each instance of this object type comprises information represented by instances of the former three object types. An object type defined in this way is called a list.

Similarly, tables can be formed by aggregations of a list type. For example, a fifth object type might also be defined in the MIB:

OBJECT:

atTable { at 1 }

Syntax:

SEQUENCE OF AtEntry

Definition:

The address translation table.

Access:

read-write.

Status:

mandatory.

such that each instance of the atTable object comprises information represented by the set of atEntry object types that collectively constitute a given atTable object instance, that is, a given address translation table.

Consider how one might refer to a simple object within a table.

Continuing with the previous example, one might name the object type

{ atPhysAddress }

and specify, using a protocol-specific mechanism, the object instance

{ atNetAddress } = { internet "10.0.0.52" }

This pairing of object type and object instance would refer to all instances of atPhysAddress which are part of any entry in some address translation table for which the associated atNetAddress value is { internet "10.0.0.52" }.

To continue with this example, consider how one might refer to an aggregate object (list) within a table. Naming the object type

{ atEntry }

and specifying, using a protocol-specific mechanism, the object instance

{ atNetAddress } = { internet "10.0.0.52" }

refers to all instances of entries in the table for which the associated atNetAddress value is { internet "10.0.0.52" }.

Each management protocol must provide a mechanism for accessing simple (non-aggregate) object types. Each management protocol specifies whether or not it supports access to aggregate object types. Further, the protocol must specify which instances are "returned" when an object type/instance pairing refers to more than one instance of a type.

To afford support for a variety of management protocols, all information by which instances of a given object type may be usefully distinguished, one from another, is represented by instances of object types defined in the MIB.

RFC-1155 Structure and Identification of Management Information - Managed Objects

Macros for Managed Objects

In order to facilitate the use of tools for processing the definition of the MIB, the OBJECT-TYPE macro may be used. This macro permits the key aspects of an object type to be represented in a formal way.

```
OBJECT-TYPE MACRO ::=
BEGIN
    TYPE NOTATION ::= "SYNTAX" type (TYPE ObjectSyntax)
            "ACCESS" Access
            "STATUS" Status
    VALUE NOTATION ::= value (VALUE ObjectName)

    Access ::= "read-only"
            | "read-write"
            | "write-only"
            | "not-accessible"
    Status ::= "mandatory"
            | "optional"
            | "obsolete"
    END
```

Given the object types defined earlier, we might imagine the following definitions being present in the MIB:

```
atIndex OBJECT-TYPE
    SYNTAX   INTEGER
    ACCESS   read-write
    STATUS   mandatory
    ::= { atEntry 1 }


atPhysAddress OBJECT-TYPE
    SYNTAX   OCTET STRING
    ACCESS   read-write
    STATUS   mandatory
    ::= { atEntry 2 }


atNetAddress OBJECT-TYPE
    SYNTAX   NetworkAddress
    ACCESS   read-write
    STATUS   mandatory
    ::= { atEntry 3 }


atEntry OBJECT-TYPE
    SYNTAX   AtEntry
    ACCESS   read-write
    STATUS   mandatory
    ::= { atTable 1 }


atTable OBJECT-TYPE
```

```
        SYNTAX   SEQUENCE OF AtEntry
        ACCESS   read-write
        STATUS   mandatory
        ::= { at 1 }

    AtEntry ::= SEQUENCE {
        atIndex
            INTEGER,
        atPhysAddress
            OCTET STRING,
        atNetAddress
            NetworkAddress
        }
```

The first five definitions describe object types, relating, for example, the OBJECT DESCRIPTOR atIndex to the OBJECT IDENTIFIER { atEntry 1 }. In addition, the syntax of this object is defined (INTEGER) along with the access permitted (read-write) and status (mandatory). The sixth definition describes an ASN.1 type called AtEntry.

Extensions to the MIB

Every Internet-standard MIB document obsoletes all previous such documents. The portion of a name, termed the tail, following the OBJECT IDENTIFIER

{ mgmt version-number }

used to name objects shall remain unchanged between versions. New versions may:

(1) declare old object types obsolete (if necessary), but not delete their names;

(2) augment the definition of an object type corresponding to a list by appending non-aggregate object types to the object types in the list; or,

(3) define entirely new object types.

New versions may not:

(1) change the semantics of any previously defined object without changing the name of that object.

These rules are important because they admit easier support for multiple versions of the Internet-standard MIB. In particular, the semantics associated with the tail of a name remain constant throughout different versions of the MIB. Because multiple versions of the MIB may thus coincide in "tail-space," implementations supporting multiple versions of the MIB can be vastly simplified.

However, as a consequence, a management agent might return an instance corresponding to a superset of the expected object type. Following the principle of robustness, in this exceptional case, a manager should ignore any additional information beyond the definition of the expected object type. However, the robustness principle requires that one exercise care with respect to control actions: if an instance does not have the same syntax as its expected object type, then those control actions must fail. In both the monitoring and control cases, the name of an object returned by an operation must be identical to the name requested by an operation.

RFC-1155 Structure and Identification of Management Information

Definitions

```
RFC1155-SMI DEFINITIONS ::= BEGIN

    EXPORTS -- EVERYTHING
    internet, directory, mgmt,
    experimental, private, enterprises,
    OBJECT-TYPE, ObjectName, ObjectSyntax, SimpleSyntax,
    ApplicationSyntax, NetworkAddress, IpAddress,
    Counter, Gauge, TimeTicks, Opaque;

-- the path to the root

    internet      OBJECT IDENTIFIER ::= { iso org(3) dod(6) 1 }
    directory     OBJECT IDENTIFIER ::= { internet 1 }
    mgmt          OBJECT IDENTIFIER ::= { internet 2 }
    experimental      OBJECT IDENTIFIER ::= { internet 3 }
    private       OBJECT IDENTIFIER ::= { internet 4 }
    enterprises  OBJECT IDENTIFIER ::= { private 1 }

 -- definition of object types

    OBJECT-TYPE MACRO ::=
    BEGIN
        TYPE NOTATION ::= "SYNTAX" type (TYPE ObjectSyntax)
                          "ACCESS" Access
                          "STATUS" Status
        VALUE NOTATION ::= value (VALUE ObjectName)

        Access ::= "read-only"
                 | "read-write"
                 | "write-only"
                 | "not-accessible"
        Status ::= "mandatory"
                 | "optional"
                 | "obsolete"
    END

-- names of objects in the MIB

    ObjectName ::=
        OBJECT IDENTIFIER

         -- syntax of objects in the MIB

        ObjectSyntax ::=
             CHOICE {
                 simple
                     SimpleSyntax,

        -- note that simple SEQUENCEs are not directly
        -- mentioned here to keep things simple (i.e.,
        -- prevent mis-use).   However, application-wide
        -- types which are IMPLICITly encoded simple
        -- SEQUENCEs may appear in the following CHOICE
```

```
    application-wide
        ApplicationSyntax
}

SimpleSyntax ::=
    CHOICE {
        number
            INTEGER,
        string
            OCTET STRING,
        object
            OBJECT IDENTIFIER,
        empty
            NULL
    }

ApplicationSyntax ::=
    CHOICE {
        address
            NetworkAddress,
        counter
            Counter,
        gauge
            Gauge,
        ticks
            TimeTicks,
        arbitrary
            Opaque
-- other application-wide types, as they are
-- defined, will be added here
    }

-- application-wide types

NetworkAddress ::=
    CHOICE {
        internet
            IpAddress
    }

IpAddress ::=
    [APPLICATION 0]             -- in network-byte order
            IMPLICIT OCTET STRING (SIZE (4))

Counter ::=
    [APPLICATION 1]
            IMPLICIT INTEGER (0..4294967295)

Gauge ::=
    [APPLICATION 2]
            IMPLICIT INTEGER (0..4294967295)

TimeTicks ::=
    [APPLICATION 3]
            IMPLICIT INTEGER (0..4294967295)
```

```
    Opaque ::=
        [APPLICATION 4]              -- arbitrary ASN.1 value,
                IMPLICIT OCTET STRING     --     "double-wrapped"

END
```

Acknowledgements

This memo was influenced by three sets of contributors to earlier drafts:

First, Lee Labarre of the MITRE Corporation, who as author of the NETMAN SMI [4], presented the basic roadmap for the SMI.

Second, several individuals who provided valuable comments on this memo prior to its initial distribution:

James R. Davin, Proteon
Mark S. Fedor, NYSERNet
Craig Partridge, BBN Laboratories
Martin Lee Schoffstall, Rensselaer Polytechnic Institute
Wengyik Yeong, NYSERNet

Third, the IETF MIB working group:

Karl Auerbach, Epilogue Technology
K. Ramesh Babu, Excelan
Lawrence Besaw, Hewlett-Packard
Jeffrey D. Case, University of Tennessee at Knoxville
James R. Davin, Proteon
Mark S. Fedor, NYSERNet
Robb Foster, BBN
Phill Gross, The MITRE Corporation
Bent Torp Jensen, Convergent Technology
Lee Labarre, The MITRE Corporation
Dan Lynch, Advanced Computing Environments
Keith McCloghrie, The Wollongong Group
Dave Mackie, 3Com/Bridge
Craig Partridge, BBN (chair)
Jim Robertson, 3Com/Bridge
Marshall T. Rose, The Wollongong Group
Greg Satz, cisco
Martin Lee Schoffstall, Rensselaer Polytechnic Institute
Lou Steinberg, IBM
Dean Throop, Data General
Unni Warrier, Unisys

Authors' Addresses

Marshall T. Rose
PSI, Inc.
PSI California Office
P.O. Box 391776
Mountain View, CA 94039

Phone: (415) 961-3380
EMail: mrose@PSI.COM

Keith McCloghrie
The Wollongong Group
1129 San Antonio Road
Palo Alto, CA 04303

Phone: (415) 962-7160
EMail: sytek!kzm@HPLABS.HP.COM

8. References

[1] Information processing systems - Open Systems Interconnection, "Specification of Abstract Syntax Notation One (ASN.1)", International Organization for Standardization, International Standard 8824, December 1987.

[2] McCloghrie K., and M. Rose, "Management Information Base for Network Management of TCP/IP-based Internets", RFC 1156, Performance Systems International and Hughes LAN Systems, May 1990.

[3] Case, J., M. Fedor, M. Schoffstall, and J. Davin, The Simple Network Management Protocol", RFC 1157, University of Tennessee at Knoxville, Performance Systems International, Performance Systems International, and the MIT Laboratory for Computer Science, May 1990.

[4] LaBarre, L., "Structure and Identification of Management Information for the Internet", Internet Engineering Task Force working note, Network Information Center, SRI International, Menlo Park, California, April 1988.

[5] Cerf, V., "IAB Recommendations for the Development of Internet Network Management Standards", RFC 1052, IAB, April 1988.

[6] Cerf, V., "Report of the Second Ad Hoc Network Management Review Group", RFC 1109, IAB, August 1989.

[7] Information processing systems - Open Systems Interconnection, "Specification of Basic Encoding Rules for Abstract Notation One (ASN.1)", International Organization for Standardization, International Standard 8825, December 1987.

Security Considerations

Security issues are not discussed in this memo.

RFC-1156 Management Information Base for Network Management of TCP/IP-based Internets

Keith McCloghrie & Marshall Rose

May 1990

Status of this Memo

This RFC is a re-release of RFC 1066, with a changed "Status of this Memo", "IAB Policy Statement", and "Introduction" sections plus a few minor typographical corrections. The technical content of the document is unchanged from RFC 1066.

This memo provides the initial version of the Management Information Base (MIB) for use with network management protocols in TCP/IP-based internets in the short-term. In particular, together with its companion memos which describe the structure of management information along with the initial network management protocol, these documents provide a simple, workable architecture and system for managing TCP/IP-based internets and in particular the Internet.

This memo specifies a Standard Protocol for the Internet community. TCP/IP implementations in the Internet which are network manageable are expected to adopt and implement this specification.

The Internet Activities Board recommends that all IP and TCP implementations be network manageable. This implies implementation of the Internet MIB (RFC-1156) and at least one of the two recommended management protocols SNMP (RFC-1157) or CMOT (RFC-1095). It should be noted that, at this time, SNMP is a full Internet standard and CMOT is a draft standard. See also the Host and Gateway Requirements RFCs for more specific information on the applicability of this standard.

Please refer to the latest edition of the "IAB Official Protocol Standards" RFC for current information on the state and status of standard Internet protocols.

Distribution of this memo is unlimited.

IAB Policy Statement

This MIB specification is the first edition of an evolving document defining variables needed for monitoring and control of various components of the Internet. Not all groups of defined variables are mandatory for all Internet components.

For example, the EGP group is mandatory for gateways using EGP but not for hosts which should not be running EGP. Similarly, the TCP group is mandatory for hosts running TCP but not for gateways which aren't running it. What IS mandatory, however, is that all variables of a group be supported if any element of the group is supported.

It is expected that additional MIB groups and variables will be defined over time to accommodate the monitoring and control needs of new or changing components of the Internet. The responsible working group(s) will continue to refine this specification.

RFC-1156 Management Information Base for Network Management

Introduction

As reported in RFC 1052, IAB Recommendations for the Development of Internet Network Management Standards [1], the Internet Activities Board has directed the Internet Engineering Task Force (IETF) to create two new working groups in the area of network management. One group was charged with the further specification and definition of elements to be included in the Management Information Base. The other was charged with defining the modifications to the Simple Network Management Protocol (SNMP) to accommodate the short-term needs of the network vendor and operator communities. In the long- term, the use of the OSI network management framework was to be examined using the ISO CMIS/CMIP [2,3] framework as a basis. Two documents were produced to define the management information: RFC-1065, which defined the Structure of Management Information (SMI), and RFC-1066, which defined the Management Information Base (MIB). Both of these documents were designed so as to be compatible with both the SNMP and the OSI network management framework.

This strategy was quite successful in the short-term: Internet-based network management technology was fielded, by both the research and commercial communities, within a few months. As a result of this, portions of the Internet community became network manageable in a timely fashion.

As reported in RFC 1109, Report of the Second Ad Hoc Network Management Review Group [6], the requirements of the SNMP and the OSI network management frameworks were more different than anticipated. As such, the requirement for compatibility between the SMI/MIB and both frameworks was suspended.

The IAB has designated the SNMP, SMI, and the initial Internet MIB to be full "Standard Protocols" with "Recommended" status. By this action, the IAB recommends that all IP and TCP implementations be network manageable and that the implementations that are network manageable are expected to adopt and implement the SMI, MIB, and SNMP.

As such, the current network management framework for TCP/IP- based internets consists of: Structure and Identification of Management Information for TCP/IP-based Internets, which describes how managed objects contained in the MIB are defined as set forth in RFC-1155; Management Information Base for Network Management of TCP/IP- based Internets, which describes the managed objects contained in the MIB as set forth in this memo; and, the Simple Network Management Protocol, which defines the protocol used to manage these objects, as set forth in RFC-1157.

The IAB also urged the working groups to be "extremely sensitive to the need to keep SNMP simple," and recommends that the MIB working group take as its starting inputs the MIB definitions found in the High-Level Entity Management Systems (HEMS) RFC 1024 [9], the initial SNMP specification [10], and the CMIS/CMIP memos [11,12].

Thus, the list of managed objects defined here, has been derived by taking only those elements which are considered essential. Since such elements are essential, there is no need to allow the implementation of individual objects, to be optional. Rather, all compliant implementations will contain all applicable (see below) objects defined in this memo.

This approach of taking only the essential objects is NOT restrictive, since the SMI defined in the companion memo provides three extensibility mechanisms: one, the addition of new standard objects through the definitions of new versions of the MIB; two, the addition of widely-available but non-standard objects through the multilateral subtree; and three, the addition of private objects through the enterprises subtree. Such additional objects can not only be used for vendor-specific elements, but also for experimentation as required to

further the knowledge of which other objects are essential.

The primary criterion for being considered essential was for an object to be contained in all of the above referenced MIB definitions. A few other objects have been included, but only if the MIB working group believed they are truly essential. The detailed list of criteria against which potential inclusions in this (initial) MIB were considered, was:

1) An object needed to be essential for either fault or configuration management.
2) Only weak control objects were permitted (by weak, it is meant that tampering with them can do only limited damage). This criterion reflects the fact that the current management protocols are not sufficiently secure to do more powerful control operations.
3) Evidence of current use and utility was required.
4) An attempt was made to limit the number of objects to about 100 to make it easier for vendors to fully instrument their software.
5) To avoid redundant variables, it was required that no object be included that can be derived from others in the MIB.
6) Implementation specific objects (e.g., for BSD UNIX)

were excluded.

7) It was agreed to avoid heavily instrumenting critical sections of code. The general guideline was one counter per critical section per layer.

Objects

Managed objects are accessed via a virtual information store, termed the Management Information Base or MIB. Objects in the MIB are defined using Abstract Syntax Notation One (ASN.1) [13].

The mechanisms used for describing these objects are specified in the companion memo. In particular, each object has a name, a syntax, and an encoding. The name is an object identifier, an administratively assigned name, which specifies an object type. The object type together with an object instance serves to uniquely identify a specific instantiation of the object. For human convenience, we often use a textual string, termed the OBJECT DESCRIPTOR, to also refer to the object type.

The syntax of an object type defines the abstract data structure corresponding to that object type. The ASN.1 language is used for this purpose. However, the companion memo purposely restricts the ASN.1 constructs which may be used. These restrictions are explicitly made for simplicity.

The encoding of an object type is simply how that object type is represented using the object type's syntax. Implicitly tied to the notion of an object type's syntax and encoding is how the object type is represented when being transmitted on the network. This memo specifies the use of the basic encoding rules of ASN.1 [14].

Object Groups
Format of Definitions

Object Groups

Since this list of managed objects contains only the essential elements, there is no need to allow individual objects to be optional. Rather, the objects are arranged into the following groups:

System
Interfaces
Address Translation
IP
ICMP
TCP
UDP
EGP

There are two reasons for defining these groups: one, to provide a means of assigning object identifiers; two, to provide a method for implementations of managed agents to know which objects they must implement. This method is as follows: if the semantics of a group is applicable to an implementation, then it must implement all objects in that group. For example, an implementation must implement the EGP group if and only if it implements the EGP protocol.

Format of Definitions

The next section contains the specification of all object types contained in the MIB. Following the conventions of the companion memo, the object types are defined using the following fields:

OBJECT:

A textual name, termed the OBJECT DESCRIPTOR, for the object type, along with its corresponding OBJECT IDENTIFIER.

Syntax:

The abstract syntax for the object type, presented using ASN.1. This must resolve to an instance of the ASN.1 type ObjectSyntax defined in the SMI.

Definition:

A textual description of the semantics of the object type. Implementations should ensure that their interpretation of the object type fulfills this definition since this MIB is intended for use in multi- vendor environments. As such it is vital that object types have consistent meaning across all machines.

Access:

One of read-only, read-write, write-only, or not-accessible.

Status:

One of mandatory, optional, or obsolete.

RFC-1156 Management Information Base for Network Management

Object Definitions


```
RFC1156-MIB

DEFINITIONS ::= BEGIN
    IMPORTS
    mgmt, OBJECT-TYPE, NetworkAddress, IpAddress,
    Counter, Gauge, TimeTicks
        FROM RFC1155-SMI;
            mib         OBJECT IDENTIFIER ::= { mgmt 1 }
            system      OBJECT IDENTIFIER ::= { mib 1 }
            interfaces  OBJECT IDENTIFIER ::= { mib 2 }
            at          OBJECT IDENTIFIER ::= { mib 3 }
            ip          OBJECT IDENTIFIER ::= { mib 4 }
            icmp        OBJECT IDENTIFIER ::= { mib 5 }
            tcp         OBJECT IDENTIFIER ::= { mib 6 }
            udp         OBJECT IDENTIFIER ::= { mib 7 }
            egp         OBJECT IDENTIFIER ::= { mib 8 }
END
```

RFC-1156 Management Information Base for Network Management - Object Definitions

The System Group

Implementation of the System group is mandatory for all systems.

sysDescr
sysObjectID
sysUpTime

RFC-1156 Management Information Base for Network Management - System Group

OBJECT: **sysDescr** { system 1 }

Syntax:
OCTET STRING

Definition:
A textual description of the entity. This value should include the full name and version identification of the system's hardware type, software operating-system, and networking software. It is mandatory that this only contain printable ASCII characters.

Access:
read-only.

Status:
mandatory.

RFC-1156 Management Information Base for Network Management - System Group

OBJECT: **sysObjectID** { system 2 }

Syntax:
OBJECT IDENTIFIER

Definition:
The vendor's authoritative identification of the network management subsystem contained in the entity. This value is allocated within the SMI enterprises subtree (1.3.6.1.4.1) and provides an easy and unambiguous means for determining "what kind of box" is being managed. For example, if vendor "Flintstones, Inc." was assigned the subtree 1.3.6.1.4.1.42, it could assign the identifier 1.3.6.1.4.1.42.1.1 to its "Fred Router".

Access:
read-only.

Status:
mandatory.

RFC-1156 Management Information Base for Network Management - System Group

OBJECT: **sysUpTime** { system 3 }

Syntax:
TimeTicks

Definition:
The time (in hundredths of a second) since the network management portion of the system was last re-initialized.

Access:
read-only.

Status:
mandatory.

The Interfaces Group

Implementation of the Interfaces group is mandatory for all systems. See also Interfaces Table.

OBJECT:
ifNumber { interfaces 1 }
Syntax:
INTEGER
Definition:
The number of network interfaces (regardless of their current state) on which this system can send/receive IP datagrams.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base for Network Management - Object Definitions

The Interfaces Table

Administrative Objects:

ifTable	**ifSpeed**
ifEntry	**ifPhysAddress**
ifIndex	**ifAdminStatus**
ifDescr	**ifOperStatus**
ifType	**ifLastChange**
ifMTU	

Input	Output
ifInOctets	**ifOutOctets**
ifInNUcastPkts	**ifOutUcastPkts**
ifInDiscards	**ifOutNUcastPkts**
ifInErrors	**ifOutDiscards**
ifInUnknownProtos	**ifOutQLen**

RFC-1156 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifTable** { interfaces 2 }

Syntax:
SEQUENCE OF IfEntry

Definition:
A list of interface entries. The number of entries is given by the value of ifNumber.

Access:
read-write.

Status:
mandatory.

RFC-1156 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifEntry** { ifTable 1 }

Syntax:

```
IfEntry ::= SEQUENCE {
    ifIndex
        INTEGER,
    ifDescr
        OCTET STRING,
    ifType
        INTEGER,
    ifMtu
        INTEGER,
    ifSpeed
        Gauge,
    ifPhysAddress
        OCTET STRING,
    ifAdminStatus
        INTEGER,
    ifOperStatus
        INTEGER,
    ifLastChange
        TimeTicks,
    ifInOctets
        Counter,
    ifInUcastPkts
        Counter,
    ifInNUcastPkts
        Counter,
    ifInDiscards
        Counter,
    ifInErrors
        Counter,
    ifInUnknownProtos
        Counter,
    ifOutOctets
        Counter,
    ifOutUcastPkts
        Counter,
    ifOutNUcastPkts
        Counter,
    ifOutDiscards
        Counter,
    ifOutErrors
        Counter,
    ifOutQLen
        Gauge
}
```

Definition:

An interface entry containing objects at the subnetwork layer and below for a particular interface.

Access:

read-write.

Status:

mandatory.

RFC-1156 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifIndex** { ifEntry 1 }

Syntax:
INTEGER

Definition:
A unique value for each interface. Its value ranges between 1 and the value of ifNumber. The value for each interface must remain constant at least from one re- initialization of the entity's network management system to the next re-initialization.

Access:
read-only.

Status:
mandatory.

RFC-1156 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifDescr** { ifEntry 2 }

Syntax:
OCTET STRING

Definition:
A text string containing information about the interface. This string should include the name of the manufacturer, the product name and the version of the hardware interface. The string is intended for presentation to a human; it must not contain anything but printable ASCII characters.

Access:
read-only.

Status:
mandatory.

RFC-1156 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifType** { ifEntry 3 }

Syntax:

```
INTEGER {
    other(1),                   -- none of the following
    regular1822(2),
    hdh1822(3),
    ddn-x25(4),
    rfc877-x25(5),
    ethernet-csmacd(6),
    iso88023-csmacd(7),
    iso88024-tokenBus(8),
    iso88025-tokenRing(9),
    iso88026-man(10),
    starLan(11),
    proteon-10MBit(12),
    proteon-80MBit(13),
    hyperchannel(14),
    fddi(15),
    lapb(16),
    sdlc(17),
    t1-carrier(18),
    cept(19),                   -- european equivalent of T-1
    basicIsdn(20),
    primaryIsdn(21),
                                -- proprietary serial
    propPointToPointSerial(22)
}
```

Definition:

The type of interface, distinguished according to the physical/link/network protocol(s) immediately "below" IP in the protocol stack.

Access:

read-only.

Status:

mandatory.

RFC-1156 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifMtu** { ifEntry 4 }

Syntax:
INTEGER

Definition:
The size of the largest IP datagram which can be sent/received on the interface, specified in octets.

Access:
read-only.

Status:
mandatory.

RFC-1156 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifSpeed** { ifEntry 5 }

Syntax:
Gauge

Definition:
An estimate of the interface's current bandwidth in bits per second. For interfaces which do not vary in bandwidth or for those where no accurate estimation can be made, this object should contain the nominal bandwidth.

Access:
read-only.

Status:
mandatory.

RFC-1156 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifPhysAddress** { ifEntry 6 }

Syntax:
OCTET STRING

Definition:
The interface's address at the protocol layer immediately "below" IP in the protocol stack. For interfaces which do not have such an address (e.g., a serial line), this object should contain an octet string of zero length.

Access:
read-only.

Status:
mandatory.

RFC-1156 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifAdminStatus** { ifEntry 7 }

Syntax:

```
INTEGER {
    up(1),       -- ready to pass packets
    down(2),
    testing(3)  -- in some test mode
}
```

Definition:

The desired state of the interface. The testing(3) state indicates that no operational packets can be passed.

Access:

read-write.

Status:

mandatory.

RFC-1156 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifOperStatus** { ifEntry 8 }

Syntax:

```
INTEGER {
    up(1),          -- ready to pass packets
    down(2),
    testing(3)      -- in some test mode
}
```

Definition:

The current operational state of the interface. The testing(3) state indicates that no operational packets can be passed.

Access:

read-only.

Status:

mandatory.

RFC-1156 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifLastChange** { ifEntry 9 }

Syntax:
TimeTicks

Definition:
The value of sysUpTime at the time the interface entered its current operational state. If the current state was entered prior to the last re-initialization of the local network management subsystem, then this object contains a zero value.

Access:
read-only.

Status:
mandatory.

RFC-1156 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifInOctets** { ifEntry 10 }

Syntax:
Counter
Definition:
The total number of octets received on the interface, including framing characters.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifInUcastPkts** { ifEntry 11 }

Syntax:
Counter
Definition:
The number of (subnet) unicast packets delivered to a higher-layer protocol.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifInNUcastPkts** { ifEntry 12 }

Syntax:
Counter
Definition:
The number of non-unicast (i.e., subnet broadcast or subnet multicast) packets delivered to a higher-layer protocol.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifInDiscards** { ifEntry 13 }

Syntax:
Counter

Definition:
The number of inbound packets which were chosen to be discarded even though no errors had been detected to prevent their being deliverable to a higher-layer protocol. One possible reason for discarding such a packet could be to free up buffer space.

Access:
read-only.

Status:
mandatory.

RFC-1156 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifInErrors** { ifEntry 14 }

Syntax:
Counter
Definition:
The number of inbound packets that contained errors preventing them from being deliverable to a higher-layer protocol.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifInUnknownProtos** { ifEntry 15 }

Syntax:
Counter

Definition:
The number of packets received via the interface which were discarded because of an unknown or unsupported protocol.

Access:
read-only.

Status:
mandatory.

RFC-1156 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifOutOctets** { ifEntry 16 }

Syntax:
Counter
Definition:
The total number of octets transmitted out of the interface, including framing characters.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifOutUcastPkts** { ifEntry 17 }

Syntax:
Counter

Definition:
The total number of packets that higher-level protocols requested be transmitted to a subnet-unicast address, including those that were discarded or not sent.

Access:
read-only.

Status:
mandatory.

RFC-1156 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifOutNUcastPkts** { ifEntry 18 }

Syntax:
Counter
Definition:
The total number of packets that higher-level protocols requested be transmitted to a non-unicast (i.e., a subnet broadcast or subnet multicast) address, including those that were discarded or not sent.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifOutDiscards** { ifEntry 19 }

Syntax:
Counter

Definition:
The number of outbound packets which were chosen to be discarded even though no errors had been detected to prevent their being transmitted. One possible reason for discarding such a packet could be to free up buffer space.

Access:
read-only.

Status:
mandatory.

RFC-1156 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifOutErrors** { ifEntry 20 }

Syntax:
Counter
Definition:
The number of outbound packets that could not be transmitted because of errors.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifOutQLen** { ifEntry 21 }

Syntax:
Gauge
Definition:
The length of the output packet queue (in packets).
Access:
read-only.
Status:
mandatory.

The Address Translation Group

Implementation of the Address Translation group is mandatory for all systems.

The Address Translation group contains one table which is the union across all interfaces of the translation tables for converting a NetworkAddress (e.g., an IP address) into a subnetwork-specific address. For lack of a better term, this document refers to such a subnetwork-specific address as a "physical" address.

Examples of such translation tables are: for broadcast media where ARP is in use, the translation table is equivalent to the ARP cache; or, on an X.25 network where non-algorithmic translation to X.121 addresses is required, the translation table contains the NetworkAddress to X.121 address equivalences.

RFC-1156 Management Information Base - Address Translation Group

OBJECT: **atTable** { at 1 }

Syntax:
SEQUENCE OF AtEntry

Definition:
The Address Translation tables contain the NetworkAddress to "physical" address equivalences. Some interfaces do not use translation tables for determining address equivalences (e.g., DDN-X.25 has an algorithmic method); if all interfaces are of this type, then the Address Translation table is empty, i.e., has zero entries.

Access:
read-write.

Status:
mandatory.

RFC-1156 Management Information Base - Address Translation Group

OBJECT: **atEntry** { atTable 1 }

Syntax:

```
AtEntry ::= SEQUENCE {
    atIfIndex
        INTEGER,
    atPhysAddress
        OCTET STRING,
    atNetAddress
        NetworkAddress
}
```

Definition:

Each entry contains one NetworkAddress to "physical" address equivalence.

Access:

read-write.

Status:

mandatory.

RFC-1156 Management Information Base - Address Translation Group

OBJECT: **atIfIndex** { atEntry 1 }

Syntax:
INTEGER
Definition:
The interface on which this entry's equivalence is effective. The interface identified by a particular value of this index is the same interface as identified by the same value of ifIndex.
Access:
read-write.
Status:
mandatory.

RFC-1156 Management Information Base - Address Translation Group

OBJECT: **atPhysAddress** { atEntry 2 }

Syntax:
OCTET STRING
Definition:
The media-dependent "physical" address.
Access:
read-write.
Status:
mandatory.

RFC-1156 Management Information Base - Address Translation Group

OBJECT: **atNetAddress** { atEntry 3 }

Syntax:
NetworkAddress
Definition:
The NetworkAddress (e.g., the IP address) corresponding to the media-dependent "physical" address.
Access:
read-write.
Status:
mandatory.

RFC-1156 Management Information Base for Network Management - Object Definitions

The IP Group

Implementation of the IP group is mandatory for all systems.

ipForwarding
ipDefaultTTL
ipInReceives
ipInHdrErrors
ipInAddrErrors
ipForwDatagrams
ipInUnknownProtos
ipInDiscards
ipInDelivers
ipOutRequests
ipOutDiscards
ipOutNoRoutes
ipReasmTimeout
ipReasmReqds
ipReasmOKs
ipReasmFails
ipFragOKs
ipFragFails
ipFragCreates

RFC-1156 Management Information Base - IP Group

OBJECT: **ipForwarding** { ip 1 }

Syntax:

```
INTEGER {
    gateway(1),     -- entity forwards datagrams
    host(2)         -- entity does NOT forward datagrams
}
```

Definition:

The indication of whether this entity is acting as an IP gateway in respect to the forwarding of datagrams received by, but not addressed to, this entity. IP gateways forward datagrams; Hosts do not (except those Source-Routed via the host).

Access:

read-only.

Status:

mandatory.

RFC-1156 Management Information Base - IP Group

OBJECT: **ipDefaultTTL** { ip 2 }

Syntax:
INTEGER

Definition:
The default value inserted into the Time-To-Live field of the IP header of datagrams originated at this entity, whenever a TTL value is not supplied by the transport layer protocol.

Access:
read-write.

Status:
mandatory.

RFC-1156 Management Information Base - IP Group

OBJECT: **ipInReceives** { ip 3 }

Syntax:
Counter
Definition:
The total number of input datagrams received from interfaces, including those received in error.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - IP Group

OBJECT: **ipInHdrErrors** { ip 4 }

Syntax:
Counter

Definition:
The number of input datagrams discarded due to errors in their IP headers, including bad checksums, version number mismatch, other format errors, time-to-live exceeded, errors discovered in processing their IP options, etc.

Access:
read-only.

Status:
mandatory.

RFC-1156 Management Information Base - IP Group

OBJECT: **ipInAddrErrors** { ip 5 }

Syntax:
Counter

Definition:
The number of input datagrams discarded because the IP address in their IP header's destination field was not a valid address to be received at this entity. This count includes invalid addresses (e.g., 0.0.0.0) and addresses of unsupported Classes (e.g., Class E). For entities which are not IP Gateways and therefore do not forward datagrams, this counter includes datagrams discarded because the destination address was not a local address.

Access:
read-only.

Status:
mandatory.

RFC-1156 Management Information Base - IP Group

OBJECT: **ipForwDatagrams** { ip 6 }

Syntax:
Counter

Definition:
The number of input datagrams for which this entity was not their final IP destination, as a result of which an attempt was made to find a route to forward them to that final destination. In entities which do not act as IP Gateways, this counter will include only those packets which were Source-Routed via this entity, and the Source-Route option processing was successful.

Access:
read-only.

Status:
mandatory.

RFC-1156 Management Information Base - IP Group

OBJECT: **ipInUnknownProtos** { ip 7 }

Syntax:
Counter
Definition:
The number of locally-addressed datagrams received successfully but discarded because of an unknown or unsupported protocol.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - IP Group

OBJECT: **ipInDiscards** { ip 8 }

Syntax:
Counter
Definition:
The number of input IP datagrams for which no problems were encountered to prevent their continued processing, but which were discarded (e.g. for lack of buffer space). Note that this counter does not include any datagrams discarded while awaiting re-assembly.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - IP Group

OBJECT: **ipInDelivers** { ip 9 }

Syntax:
Counter
Definition:
The total number of input datagrams successfully delivered to IP user-protocols (including ICMP).
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - IP Group

OBJECT: **ipOutRequests** { ip 10 }

Syntax:
Counter

Definition:
The total number of IP datagrams which local IP user- protocols (including ICMP) supplied to IP in requests for transmission. Note that this counter does not include any datagrams counted in ipForwDatagrams.

Access:
read-only.

Status:
mandatory.

RFC-1156 Management Information Base - IP Group

OBJECT: **ipOutDiscards** { ip 11 }

Syntax:
Counter

Definition:
The number of output IP datagrams for which no problem was encountered to prevent their transmission to their destination, but which were discarded (e.g., for lack of buffer space). Note that this counter would include datagrams counted in ipForwDatagrams if any such packets met this (discretionary) discard criterion.

Access:
read-only.

Status:
mandatory.

RFC-1156 Management Information Base - IP Group

OBJECT: **ipOutNoRoutes** { ip 12 }

Syntax:
Counter

Definition:
The number of IP datagrams discarded because no route could be found to transmit them to their destination. Note that this counter includes any packets counted in ipForwDatagrams which meet this "no-route" criterion.

Access:
read-only.

Status:
mandatory.

RFC-1156 Management Information Base - IP Group

OBJECT: **ipReasmTimeout** { ip 13 }

Syntax:
INTEGER
Definition:
The maximum number of seconds which received fragments are held while they are awaiting reassembly at this entity.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - IP Group

OBJECT: **ipReasmReqds** { ip 14 }

Syntax:
Counter
Definition:
The number of IP fragments received which needed to be reassembled at this entity.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - IP Group

OBJECT: **ipReasmOKs** { ip 15 }

Syntax:
Counter
Definition:
The number of IP datagrams successfully re-assembled.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - IP Group

OBJECT: **ipReasmFails** { ip 16 }

Syntax:
Counter

Definition:
The number of failures detected by the IP re-assembly algorithm (for whatever reason: timed out, errors, etc).
Note that this is not necessarily a count of discarded IP fragments since some algorithms (notably RFC 815's) can lose track of the number of fragments by combining them as they are received.

Access:
read-only.

Status:
mandatory.

RFC-1156 Management Information Base - IP Group

OBJECT: **ipFragOKs** { ip 17 }

Syntax:
Counter
Definition:
The number of IP datagrams that have been successfully fragmented at this entity.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - IP Group

OBJECT: **ipFragFails** { ip 18 }

Syntax:
Counter

Definition:
The number of IP datagrams that have been discarded because they needed to be fragmented at this entity but could not be, e.g., because their "Don't Fragment" flag was set.

Access:
read-only.

Status:
mandatory.

OBJECT: **ipFragCreates** { ip 19 }

Syntax:
Counter
Definition:
The number of IP datagram fragments that have been generated as a result of fragmentation at this entity.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base for Network Management - Object Definitions

The IP Address Table

The Ip Address table contains this entity's IP addressing information.

ipAddrTable
ipAddrEntry
ipAdEntAddr
ipAdEntIfIndex
ipAdEntNetMask
ipAdEntBcastAddr

RFC-1156 Management Information Base - IP Address Table

OBJECT: **ipAddrTable** { ip 20 }

Syntax:
SEQUENCE OF IpAddrEntry
Definition:
The table of addressing information relevant to this entity's IP addresses.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - IP Address Table

OBJECT: **ipAddrEntry** { ipAddrTable 1 }

Syntax:

```
IpAddrEntry ::= SEQUENCE {
    ipAdEntAddr
        IpAddress,
    ipAdEntIfIndex
        INTEGER,
    ipAdEntNetMask
        IpAddress,
    ipAdEntBcastAddr
        INTEGER
}
```

Definition:

The addressing information for one of this entity's IP addresses.

Access:

read-only.

Status:

mandatory.

RFC-1156 Management Information Base - IP Address Table

OBJECT: **ipAdEntAddr** { ipAddrEntry 1 }

Syntax:
IpAddress
Definition:
The IP address to which this entry's addressing information pertains.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - IP Address Table

OBJECT: **ipAdEntIfIndex** { ipAddrEntry 2 }

Syntax:
INTEGER
Definition:
The index value which uniquely identifies the interface to which this entry is applicable. The interface identified by a particular value of this index is the same interface as identified by the same value of ifIndex.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - IP Address Table

OBJECT: **ipAdEntNetMask** { ipAddrEntry 3 }

Syntax:
IpAddress
Definition:
The subnet mask associated with the IP address of this entry. The value of the mask is an IP address with all the network bits set to 1 and all the hosts bits set to 0.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - IP Address Table

OBJECT: **ipAdEntBcastAddr** { ipAddrEntry 4 }

Syntax:
INTEGER
Definition:
The value of the least-significant bit in the IP broadcast address used for sending datagrams on the (logical) interface associated with the IP address of this entry. For example, when the Internet standard all-ones broadcast address is used, the value will be 1.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base for Network Management - Object Definitions

The IP Routing Table

The IP Routing Table contains an entry for each route presently known to this entity. Note that the action to be taken in response to a request to read a non-existent entry, is specific to the network management protocol being used.

RFC-1156 Management Information Base - IP Routing Table

OBJECT: **ipRoutingTable** { ip 21 }

Syntax:
SEQUENCE OF IpRouteEntry
Definition:
This entity's IP Routing table.
Access:
read-write.
Status:
mandatory.

RFC-1156 Management Information Base - IP Routing Table

OBJECT: **ipRouteEntry** { ipRoutingTable 1 }

Syntax:

```
IpRouteEntry ::= SEQUENCE {
    ipRouteDest
        IpAddress,
    ipRouteIfIndex
        INTEGER,
    ipRouteMetric1
        INTEGER,
    ipRouteMetric2
        INTEGER,
    ipRouteMetric3
        INTEGER,
    ipRouteMetric4
        INTEGER,
    ipRouteNextHop
        IpAddress,
    ipRouteType
        INTEGER,
    ipRouteProto
        INTEGER,
    ipRouteAge
        INTEGER
}
```

Definition:

A route to a particular destination.

Access:

read-write.

Status:

mandatory.

RFC-1156 Management Information Base - IP Routing Table

OBJECT: **ipRouteDest** { ipRouteEntry 1 }

Syntax:
IpAddress
Definition:
The destination IP address of this route. An entry with a value of 0.0.0.0 is considered a default route. Multiple such default routes can appear in the table, but access to such multiple entries is dependent on the table-access mechanisms defined by the network management protocol in use.
Access:
read-write.
Status:
mandatory.

RFC-1156 Management Information Base - IP Routing Table

OBJECT: **ipRouteIfIndex** { ipRouteEntry 2 }

Syntax:
INTEGER
Definition:
The index value which uniquely identifies the local interface through which the next hop of this route should be reached. The interface identified by a particular value of this index is the same interface as identified by the same value of ifIndex.
Access:
read-write.
Status:
mandatory.

RFC-1156 Management Information Base - IP Routing Table

OBJECT: **ipRouteMetric1** { ipRouteEntry 3 }

Syntax:
INTEGER
Definition:
The primary routing metric for this route. The semantics of this metric are determined by the routing-protocol specified in the route's ipRouteProto value. If this metric is not used, its value should be set to -1.
Access:
read-write.
Status:
mandatory.

RFC-1156 Management Information Base - IP Routing Table

OBJECT: **ipRouteMetric2** { ipRouteEntry 4 }

Syntax:
INTEGER
Definition:
An alternate routing metric for this route. The semantics of this metric are determined by the routing- protocol specified in the route's ipRouteProto value. If this metric is not used, its value should be set to -1.
Access:
read-write.
Status:
mandatory.

RFC-1156 Management Information Base - IP Routing Table

OBJECT: **ipRouteMetric3** { ipRouteEntry 5 }

Syntax:
INTEGER
Definition:
An alternate routing metric for this route. The semantics of this metric are determined by the routing- protocol specified in the route's ipRouteProto value. If this metric is not used, its value should be set to -1.
Access:
read-write.
Status:
mandatory.

RFC-1156 Management Information Base - IP Routing Table

OBJECT: **ipRouteMetric4** { ipRouteEntry 6 }

Syntax:
INTEGER
Definition:
An alternate routing metric for this route. The semantics of this metric are determined by the routing- protocol specified in the route's ipRouteProto value. If this metric is not used, its value should be set to -1.
Access:
read-write.
Status:
mandatory.

RFC-1156 Management Information Base - IP Routing Table

OBJECT: **ipRouteNextHop** { ipRouteEntry 7 }

Syntax:
IpAddress
Definition:
The IP address of the next hop of this route.
Access:
read-write.
Status:
mandatory.

RFC-1156 Management Information Base - IP Routing Table

OBJECT: **ipRouteType** { ipRouteEntry 8 }

Syntax:

```
INTEGER {
    other(1),   -- none of the following
    invalid(2), -- an invalidated route
    direct(3),  -- route to directly connected (sub-)network
    remote(4),-- route to a non-local host/network/sub-network
}
```

Definition:

The type of route.

Access:

read-write.

Status:

mandatory.

RFC-1156 Management Information Base - IP Routing Table

OBJECT: **ipRouteProto** { ipRouteEntry 9 }

Syntax:

```
INTEGER {
    other(1),       -- none of the following

                    -- non-protocol information,
                    -- e.g., manually configured
    local(2),       -- entries

                    -- set via a network management
    netmgmt(3),         -- protocol

                    -- obtained via ICMP,
    icmp(4),        -- e.g., Redirect

                    -- the remaining values are
                    -- all gateway routing protocols
    egp(5),
    ggp(6),
    hello(7),
    rip(8),
    is-is(9),
    es-is(10),
    ciscoIgrp(11),
    bbnSpfIgp(12),
    oigp(13)
}
```

Definition:

The routing mechanism via which this route was learned. Inclusion of values for gateway routing protocols is not intended to imply that hosts should support those protocols.

Access:

read-only.

Status:

mandatory.

RFC-1156 Management Information Base - IP Routing Table

OBJECT: **ipRouteAge** { ipRouteEntry 10 }

Syntax:
INTEGER
Definition:
The number of seconds since this route was last updated or otherwise determined to be correct. Note that no semantics of "too old" can be implied except through knowledge of the routing protocol by which the route was learned.
Access:
read-write.
Status:
mandatory.

RFC-1156 Management Information Base for Network Management - Object Definitions

The ICMP Group

Implementation of the ICMP group is mandatory for all systems.

The ICMP group contains the ICMP input and output statistics.

Note that individual counters for ICMP message (sub-)codes have been omitted from this (version of the) MIB for simplicity.

icmpInMsgs	**icmpOutMsgs**
icmpInErrors	**icmpOutErrors**
icmpInDestUnreachs	**icmpOutDestUnreachs**
icmpInTimeExcds	**icmpOutTimeExcds**
icmpInParmProbs	**icmpOutParmProbs**
icmpInSrcQuenchs	**icmpOutSrcQuenchs**
icmpInRedirects	**icmpOutRedirects**
icmpInEchos	**icmpOutEchos**
icmpInEchoReps	**icmpOutEchoReps**
icmpInTimestamps	**icmpOutTimestamps**
icmpInTimestampReps	**icmpOutTimestampReps**
icmpInAddrMasks	**icmpOutAddrMasks**
icmpInAddrMaskReps	**icmpOutAddrMaskReps**

RFC-1156 Management Information Base - ICMP Group

OBJECT: **icmpInMsgs** { icmp 1 }

Syntax:
Counter
Definition:
The total number of ICMP messages which the entity received. Note that this counter includes all those counted by icmpInErrors.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - ICMP Group

OBJECT: **icmpInErrors** { icmp 2 }

Syntax:
Counter
Definition:
The number of ICMP messages which the entity received but determined as having errors (bad ICMP checksums, bad length, etc.).
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - ICMP Group

OBJECT: **icmpInDestUnreachs** { icmp 3 }

Syntax:
Counter
Definition:
The number of ICMP Destination Unreachable messages received.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - ICMP Group

OBJECT: **icmpInTimeExcds** { icmp 4 }

Syntax:
Counter
Definition:
The number of ICMP Time Exceeded messages received.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - ICMP Group

OBJECT: **icmpInParmProbs** { icmp 5 }

Syntax:
Counter
Definition:
The number of ICMP Parameter Problem messages received.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - ICMP Group

OBJECT: **icmpInSrcQuenchs** { icmp 6 }

Syntax:
Counter
Definition:
The number of ICMP Source Quench messages received.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - ICMP Group

OBJECT: **icmpInRedirects** { icmp 7 }

Syntax:
Counter
Definition:
The number of ICMP Redirect messages received.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - ICMP Group

OBJECT: **icmpInEchos** { icmp 8 }

Syntax:
Counter
Definition:
The number of ICMP Echo (request) messages received.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - ICMP Group

OBJECT: **icmpInEchoReps** { icmp 9 }

Syntax:
Counter
Definition:
The number of ICMP Echo Reply messages received.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - ICMP Group

OBJECT: **icmpInTimestamps** { icmp 10 }

Syntax:
Counter
Definition:
The number of ICMP Timestamp (request) messages received.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - ICMP Group

OBJECT: **icmpInTimestampReps** { icmp 11 }

Syntax:
Counter
Definition:
The number of ICMP Timestamp Reply messages received.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - ICMP Group

OBJECT: **icmpInAddrMasks** { icmp 12 }

Syntax:
Counter
Definition:
The number of ICMP Address Mask Request messages received.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - ICMP Group

OBJECT: **icmpInAddrMaskReps** { icmp 13 }

Syntax:
Counter
Definition:
The number of ICMP Address Mask Reply messages received.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - ICMP Group

OBJECT: **icmpOutMsgs** { icmp 14 }

Syntax:
Counter
Definition:
The total number of ICMP messages which this entity attempted to send. Note that this counter includes all those counted by icmpOutErrors.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - ICMP Group

OBJECT: **icmpOutErrors** { icmp 15 }

Syntax:
Counter

Definition:
The number of ICMP messages which this entity did not send due to problems discovered within ICMP such as a lack of buffers. This value should not include errors discovered outside the ICMP layer such as the inability of IP to route the resultant datagram. In some implementations there may be no types of error which contribute to this counter's value.

Access:
read-only.

Status:
mandatory.

RFC-1156 Management Information Base - ICMP Group

OBJECT: **icmpOutDestUnreachs** { icmp 16 }

Syntax:
Counter
Definition:
The number of ICMP Destination Unreachable messages sent.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - ICMP Group

OBJECT: **icmpOutTimeExcds** { icmp 17 }

Syntax:
Counter
Definition:
The number of ICMP Time Exceeded messages sent.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - ICMP Group

OBJECT: **icmpOutParmProbs** { icmp 18 }

Syntax:
Counter
Definition:
The number of ICMP Parameter Problem messages sent.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - ICMP Group

OBJECT: **icmpOutSrcQuenchs** { icmp 19 }

Syntax:
Counter
Definition:
The number of ICMP Source Quench messages sent.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - ICMP Group

OBJECT: **icmpOutRedirects** { icmp 20 }

Syntax:
Counter
Definition:
The number of ICMP Redirect messages sent.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - ICMP Group

OBJECT: **icmpOutEchos** { icmp 21 }

Syntax:
Counter
Definition:
The number of ICMP Echo (request) messages sent.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - ICMP Group

OBJECT: **icmpOutEchoReps** { icmp 22 }

Syntax:
Counter
Definition:
The number of ICMP Echo Reply messages sent.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - ICMP Group

OBJECT: **icmpOutTimestamps** { icmp 23 }

Syntax:
Counter
Definition:
The number of ICMP Timestamp (request) messages sent.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - ICMP Group

OBJECT: **icmpOutTimestampReps** { icmp 24 }

Syntax:
Counter
Definition:
The number of ICMP Timestamp Reply messages sent.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - ICMP Group

OBJECT: **icmpOutAddrMasks** { icmp 25 }

Syntax:
Counter
Definition:
The number of ICMP Address Mask Request messages sent.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - ICMP Group

OBJECT: **icmpOutAddrMaskReps** { icmp 26 }

Syntax:
Counter
Definition:
The number of ICMP Address Mask Reply messages sent.
Access:
read-only.
Status:
mandatory.

The TCP Group

Implementation of the TCP group is mandatory for all systems that implement the TCP protocol.

Note that instances of object types that represent information about a particular TCP connection are transient; they persist only as long as the connection in question.

RFC-1156 Management Information Base - TCP Group

OBJECT: **tcpRtoAlgorithm** { tcp 1 }

Syntax:

```
INTEGER {
    other(1),      -- none of the following
    constant(2), -- a constant rto
    rsre(3),       -- MIL-STD-1778, Appendix B
    vanj(4)        -- Van Jacobson's algorithm [15]
}
```

Definition:

The algorithm used to determine the timeout value used for retransmitting unacknowledged octets.

Access:

read-only.

Status:

mandatory.

RFC-1156 Management Information Base - TCP Group

OBJECT: **tcpRtoMin** { tcp 2 }

Syntax:
INTEGER
Definition:
The minimum value permitted by a TCP implementation for the retransmission timeout, measured in milliseconds. More refined semantics for objects of this type depend upon the algorithm used to determine the retransmission timeout. In particular, when the timeout algorithm is rsre(3), an object of this type has the semantics of the LBOUND quantity described in RFC 793.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - TCP Group

OBJECT: **tcpRtoMax** { tcp 3 }

Syntax:
INTEGER

Definition:
The maximum value permitted by a TCP implementation for the retransmission timeout, measured in milliseconds. More refined semantics for objects of this type depend upon the algorithm used to determine the retransmission timeout. In particular, when the timeout algorithm is rsre(3), an object of this type has the semantics of the UBOUND quantity described in RFC 793.

Access:
read-only.

Status:
mandatory.

RFC-1156 Management Information Base - TCP Group

OBJECT: **tcpMaxConn** { tcp 4 }

Syntax:
INTEGER
Definition:
The limit on the total number of TCP connections the entity can support. In entities where the maximum number of connections is dynamic, this object should contain the value "-1".
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - TCP Group

OBJECT: **tcpActiveOpens** { tcp 5 }

Syntax:
Counter
Definition:
The number of times TCP connections have made a direct transition to the SYN-SENT state from the CLOSED state.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - TCP Group

OBJECT: **tcpPassiveOpens** { tcp 6 }

Syntax:
Counter
Definition:
The number of times TCP connections have made a direct transition to the SYN-RCVD state from the LISTEN state.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - TCP Group

OBJECT: **tcpAttemptFails** { tcp 7 }

Syntax:
Counter
Definition:
The number of times TCP connections have made a direct transition to the CLOSED state from either the SYN-SENT state or the SYN-RCVD state, plus the number of times TCP connections have made a direct transition to the LISTEN state from the SYN-RCVD state.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - TCP Group

OBJECT: **tcpEstabResets** { tcp 8 }

Syntax:
Counter
Definition:
The number of times TCP connections have made a direct transition to the CLOSED state from either the ESTABLISHED state or the CLOSE-WAIT state.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - TCP Group

OBJECT: **tcpCurrEstab** { tcp 9 }

Syntax:
Gauge
Definition:
The number of TCP connections for which the current state is either ESTABLISHED or CLOSE-WAIT.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - TCP Group

OBJECT: **tcpInSegs** { tcp 10 }

Syntax:
Counter
Definition:
The total number of segments received, including those received in error.
This count includes segments received on currently established connections.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - TCP Group

OBJECT: **tcpOutSegs** { tcp 11 }

Syntax:
Counter
Definition:
The total number of segments sent, including those on current connections but excluding those containing only retransmitted octets.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - TCP Group

OBJECT: **tcpRetransSegs** { tcp 12 }

Syntax:
Counter
Definition:
The total number of segments retransmitted - that is, the number of TCP segments transmitted containing one or more previously transmitted octets.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - TCP Group

OBJECT: **tcpConnTable** { tcp 13 }

Syntax:
SEQUENCE OF TcpConnEntry
Definition:
A table containing TCP connection-specific information.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - TCP Group

OBJECT: **tcpConnEntry** { tcpConnTable 1 }

Syntax:

```
TcpConnEntry ::= SEQUENCE {
    tcpConnState
        INTEGER,
    tcpConnLocalAddress
        IpAddress,
    tcpConnLocalPort
        INTEGER (0..65535),
    tcpConnRemAddress
        IpAddress,
    tcpConnRemPort
        INTEGER (0..65535)
}
```

Definition:

Information about a particular current TCP connection. An object of this type is transient, in that it ceases to exist when (or soon after) the connection makes the transition to the CLOSED state.

Access:

read-only.

Status:

mandatory.

RFC-1156 Management Information Base - TCP Group

OBJECT: **tcpConnState** { tcpConnEntry 1 }

Syntax:

INTEGER {
closed(1),
listen(2),
synSent(3),
synReceived(4),
established(5),
finWait1(6),
finWait2(7),
closeWait(8),
lastAck(9),
closing(10),
timeWait(11)
}

Definition:

The state of this TCP connection.

Access:

read-only.

Status:

mandatory.

RFC-1156 Management Information Base - TCP Group

OBJECT: **tcpConnLocalAddress** { tcpConnEntry 2 }

Syntax:
IpAddress
Definition:
The local IP address for this TCP connection.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - TCP Group

OBJECT: **tcpConnLocalPort** { tcpConnEntry 3 }

Syntax:
INTEGER (0..65535**)**
Definition:
The local port number for this TCP connection.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - TCP Group

OBJECT: **tcpConnRemAddress** { tcpConnEntry 4 }

Syntax:
IpAddress
Definition:
The remote IP address for this TCP connection.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - TCP Group

OBJECT: **tcpConnRemPort** { tcpConnEntry 5 }

Syntax:
INTEGER (0..65535)
Definition:
The remote port number for this TCP connection.
Access:
read-only.
Status:
mandatory.

The UDP Group

Implementation of the UDP group is mandatory for all systems which implement the UDP protocol.

udpInDatagrams
udpNoPorts
udpInErrors
udpOutDatagrams

RFC-1156 Management Information Base - UDP Group

OBJECT: **udpInDatagrams** { udp 1 }

Syntax:
Counter
Definition:
The total number of UDP datagrams delivered to UDP users.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - UDP Group

OBJECT: **udpNoPorts** { udp 2 }

Syntax:
Counter
Definition:
The total number of received UDP datagrams for which there was no application at the destination port.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - UDP Group

OBJECT: **udpInErrors** { udp 3 }

Syntax:
Counter
Definition:
The number of received UDP datagrams that could not be delivered for reasons other than the lack of an application at the destination port.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - UDP Group

OBJECT: **udpOutDatagrams** { udp 4 }

Syntax:
Counter
Definition:
The total number of UDP datagrams sent from this entity.
Access:
read-only.
Status:
mandatory.

The EGP Group

Implementation of the EGP group is mandatory for all systems which implement the EGP protocol.

egpInMsgs
egpInErrors
egpOutMsgs
egpOutErrors

RFC-1156 Management Information Base - EGP Group

OBJECT: **egpInMsgs** { egp 1 }

Syntax:
Counter
Definition:
The number of EGP messages received without error.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - EGP Group

OBJECT: **egpInErrors** { egp 2 }

Syntax:
Counter
Definition:
The number of EGP messages received that proved to be in error.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - EGP Group

OBJECT: **egpOutMsgs** { egp 3 }

Syntax:
Counter
Definition:
The total number of locally generated EGP messages.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - EGP Group

OBJECT: **egpOutErrors** { egp 4 }

Syntax:
Counter
Definition:
The number of locally generated EGP messages not sent due to resource limitations within an EGP entity.
Access:
read-only.
Status:
mandatory.

The EGP Neighbor Table

The Egp Neighbor table contains information about this entity's EGP neighbors.

egpNeighTable
egpNeighEntry
egpNeighState
egpNeighAddr

RFC-1156 Management Information Base - EGP Neighbor Table

OBJECT: **egpNeighTable** { egp 5 }

Syntax:
SEQUENCE OF EgpNeighEntry
Definition:
The EGP neighbor table.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base - EGP Neighbor Table

OBJECT: **egpNeighEntry** { egpNeighTable 1 }

Syntax:

```
EgpNeighEntry ::= SEQUENCE {
    egpNeighState
        INTEGER,
    egpNeighAddr
        IpAddress
}
```

Definition:

Information about this entity's relationship with a particular EGP neighbor.

Access:

read-only.

Status:

mandatory.

RFC-1156 Management Information Base - EGP Neighbor Table

OBJECT: **egpNeighState** { egpNeighEntry 1 }

Syntax:

INTEGER {
idle(1),
acquisition(2),
down(3),
up(4),
cease(5)
}

Definition:

The EGP state of the local system with respect to this entry's EGP neighbor. Each EGP state is represented by a value that is one greater than the numerical value associated with said state in RFC 904.

Access:

read-only.

Status:

mandatory.

RFC-1156 Management Information Base - EGP Neighbor Table

OBJECT: **egpNeighAddr** { egpNeighEntry 2 }

Syntax:
IpAddress
Definition:
The IP address of this entry's EGP neighbor.
Access:
read-only.
Status:
mandatory.

RFC-1156 Management Information Base for Network Management

Definitions

```
RFC1156-MIB

DEFINITIONS ::= BEGIN

IMPORTS
    mgmt, OBJECT-TYPE, NetworkAddress, IpAddress,
    Counter, Gauge, TimeTicks
        FROM RFC1155-SMI;

        mib         OBJECT IDENTIFIER ::= { mgmt 1 }

        system      OBJECT IDENTIFIER ::= { mib 1 }
        interfaces  OBJECT IDENTIFIER ::= { mib 2 }
        at          OBJECT IDENTIFIER ::= { mib 3 }
        ip          OBJECT IDENTIFIER ::= { mib 4 }
        icmp        OBJECT IDENTIFIER ::= { mib 5 }
        tcp         OBJECT IDENTIFIER ::= { mib 6 }
        udp         OBJECT IDENTIFIER ::= { mib 7 }
        egp         OBJECT IDENTIFIER ::= { mib 8 }
```

The System Group

```
sysDescr OBJECT-TYPE
        SYNTAX   OCTET STRING
        ACCESS   read-only
        STATUS   mandatory
        ::= { system 1 }

sysObjectID OBJECT-TYPE
        SYNTAX   OBJECT IDENTIFIER
        ACCESS   read-only
        STATUS   mandatory
        ::= { system 2 }

sysUpTime OBJECT-TYPE
        SYNTAX   TimeTicks
        ACCESS   read-only
        STATUS   mandatory
        ::= { system 3 }
```

The Interfaces Group

```
ifNumber OBJECT-TYPE
        SYNTAX   INTEGER
        ACCESS   read-only
        STATUS   mandatory
        ::= { interfaces 1 }
```

-- The Interfaces Table

```
ifTable OBJECT-TYPE
        SYNTAX   SEQUENCE OF IfEntry
        ACCESS   read-write
        STATUS   mandatory
        ::= { interfaces 2 }

ifEntry OBJECT-TYPE
        SYNTAX   IfEntry
        ACCESS   read-write
        STATUS   mandatory
        ::= { ifTable 1 }

IfEntry ::= SEQUENCE {
    ifIndex
        INTEGER,
    ifDescr
        OCTET STRING,
    ifType
        INTEGER,
    ifMtu
        INTEGER,
    ifSpeed
        Gauge,
    ifPhysAddress
        OCTET STRING,
    ifAdminStatus
        INTEGER,
    ifOperStatus
        INTEGER,
    ifLastChange
        TimeTicks,
    ifInOctets
        Counter,
    ifInUcastPkts
        Counter,
    ifInNUcastPkts
        Counter,
    ifInDiscards
        Counter,
    ifInErrors
        Counter,
    ifInUnknownProtos
        Counter,
```

```
    ifOutOctets
        Counter,
    ifOutUcastPkts
        Counter,
    ifOutNUcastPkts
        Counter,
    ifOutDiscards
        Counter,
    ifOutErrors
        Counter,
    ifOutQLen
        Gauge
}

ifIndex OBJECT-TYPE
        SYNTAX   INTEGER
        ACCESS   read-only
        STATUS   mandatory
        ::= { ifEntry 1 }

ifDescr OBJECT-TYPE
        SYNTAX   OCTET STRING
        ACCESS   read-only
        STATUS   mandatory
        ::= { ifEntry 2 }

ifType OBJECT-TYPE
        SYNTAX   INTEGER {
                other(1),       -- none of the following
                regular1822(2),
                hdh1822(3),
                ddn-x25(4),
                rfc877-x25(5),
                ethernet-csmacd(6),
                iso88023-csmacd(7),
                iso88024-tokenBus(8),
                iso88025-tokenRing(9),
                iso88026-man(10),
                starLan(11),
                proteon-10MBit(12),
                proteon-80MBit(13),
                hyperchannel(14),
                fddi(15),
                lapb(16),
                sdlc(17),
                t1-carrier(18),
                cept(19),
                basicIsdn(20),
                primaryIsdn(21),
                                -- proprietary serial
                propPointToPointSerial(22)
            }
        ACCESS   read-only
        STATUS   mandatory
        ::= { ifEntry 3 }
```

```
ifMtu OBJECT-TYPE
        SYNTAX   INTEGER
        ACCESS   read-only
        STATUS   mandatory
        ::= { ifEntry 4 }

ifSpeed OBJECT-TYPE
        SYNTAX   Gauge
        ACCESS   read-only
        STATUS   mandatory
        ::= { ifEntry 5 }

ifPhysAddress OBJECT-TYPE
        SYNTAX   OCTET STRING
        ACCESS   read-only
        STATUS   mandatory
        ::= { ifEntry 6 }

ifAdminStatus OBJECT-TYPE
        SYNTAX   INTEGER {
                 up(1),          -- ready to pass packets
                 down(2),
                 testing(3)      -- in some test mode
                 }
        ACCESS   read-write
        STATUS   mandatory
        ::= { ifEntry 7 }

ifOperStatus OBJECT-TYPE
        SYNTAX   INTEGER {
                 up(1),          -- ready to pass packets
                 down(2),
                 testing(3)      -- in some test mode
                 }
        ACCESS   read-only
        STATUS   mandatory
        ::= { ifEntry 8 }

ifLastChange OBJECT-TYPE
        SYNTAX   TimeTicks
        ACCESS   read-only
        STATUS   mandatory
        ::= { ifEntry 9 }

ifInOctets OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { ifEntry 10 }

ifInUcastPkts OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
```

```
        ::=   { ifEntry 11 }

ifInNUcastPkts OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { ifEntry 12 }

ifInDiscards OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { ifEntry 13 }

ifInErrors OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { ifEntry 14 }

ifInUnknownProtos OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { ifEntry 15 }

ifOutOctets OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { ifEntry 16 }

ifOutUcastPkts OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { ifEntry 17 }

ifOutNUcastPkts OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { ifEntry 18 }

ifOutDiscards OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { ifEntry 19 }

ifOutErrors OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { ifEntry 20 }
```

```
ifOutQLen OBJECT-TYPE
        SYNTAX   Gauge
        ACCESS   read-only
        STATUS   mandatory
        ::= { ifEntry 21 }
```

The Address Translation Group

```
atTable OBJECT-TYPE
        SYNTAX   SEQUENCE OF AtEntry
        ACCESS   read-write
        STATUS   mandatory
        ::= { at 1 }

atEntry OBJECT-TYPE
        SYNTAX   AtEntry
        ACCESS   read-write
        STATUS   mandatory
        ::= { atTable 1 }

AtEntry ::= SEQUENCE {
    atIfIndex
        INTEGER,
    atPhysAddress
        OCTET STRING,
    atNetAddress
        NetworkAddress
}

atIfIndex OBJECT-TYPE
        SYNTAX   INTEGER
        ACCESS   read-write
        STATUS   mandatory
        ::= { atEntry 1 }

atPhysAddress OBJECT-TYPE
        SYNTAX   OCTET STRING
        ACCESS   read-write
        STATUS   mandatory
        ::= { atEntry 2 }

atNetAddress OBJECT-TYPE
        SYNTAX   NetworkAddress
        ACCESS   read-write
        STATUS   mandatory
        ::= { atEntry 3 }
```

The IP Group

```
ipForwarding OBJECT-TYPE
        SYNTAX   INTEGER {
      gateway(1), -- entity forwards datagrams
      host(2)       -- entity does NOT forward datagrams
                }
        ACCESS   read-only
        STATUS   mandatory
        ::= { ip 1 }

ipDefaultTTL OBJECT-TYPE
        SYNTAX   INTEGER
        ACCESS   read-write
        STATUS   mandatory
        ::= { ip 2 }

ipInReceives OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { ip 3 }

ipInHdrErrors OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { ip 4 }

ipInAddrErrors OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { ip 5 }

ipForwDatagrams OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { ip 6 }

ipInUnknownProtos OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { ip 7 }

ipInDiscards OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { ip 8 }
```

```
ipInDelivers OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { ip 9 }

ipOutRequests OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { ip 10 }

ipOutDiscards OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { ip 11 }

ipOutNoRoutes OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { ip 12 }

ipReasmTimeout OBJECT-TYPE
        SYNTAX   INTEGER
        ACCESS   read-only
        STATUS   mandatory
        ::= { ip 13 }

ipReasmReqds OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { ip 14 }

ipReasmOKs OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { ip 15 }

ipReasmFails OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { ip 16 }

ipFragOKs OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { ip 17 }

ipFragFails OBJECT-TYPE
```

```
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { ip 18 }

ipFragCreates OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { ip 19 }
```

-- the IP Interface table

```
ipAddrTable OBJECT-TYPE
        SYNTAX   SEQUENCE OF IpAddrEntry
        ACCESS   read-only
        STATUS   mandatory
        ::= { ip 20 }

ipAddrEntry OBJECT-TYPE
        SYNTAX   IpAddrEntry
        ACCESS   read-only
        STATUS   mandatory
        ::= { ipAddrTable 1 }

IpAddrEntry ::= SEQUENCE {
    ipAdEntAddr
        IpAddress,
    ipAdEntIfIndex
        INTEGER,
    ipAdEntNetMask
        IpAddress,
    ipAdEntBcastAddr
        INTEGER
}

ipAdEntAddr OBJECT-TYPE
        SYNTAX   IpAddress
        ACCESS   read-only
        STATUS   mandatory
        ::=   { ipAddrEntry 1 }

ipAdEntIfIndex OBJECT-TYPE
        SYNTAX   INTEGER
        ACCESS   read-only
        STATUS   mandatory
        ::=   { ipAddrEntry 2 }

ipAdEntNetMask OBJECT-TYPE
        SYNTAX   IpAddress
        ACCESS   read-only
        STATUS   mandatory
        ::=   { ipAddrEntry 3 }

ipAdEntBcastAddr OBJECT-TYPE
```

```
        SYNTAX   INTEGER
        ACCESS   read-only
        STATUS   mandatory
        ::= { ipAddrEntry 4 }

-- the IP Routing table

ipRoutingTable OBJECT-TYPE
        SYNTAX   SEQUENCE OF IpRouteEntry
        ACCESS   read-write
        STATUS   mandatory
        ::= { ip 21 }

ipRouteEntry OBJECT-TYPE
        SYNTAX   IpRouteEntry
        ACCESS   read-write
        STATUS   mandatory
        ::= { ipRoutingTable 1 }

IpRouteEntry ::= SEQUENCE {
    ipRouteDest
        IpAddress,
    ipRouteIfIndex
        INTEGER,
    ipRouteMetric1
        INTEGER,
    ipRouteMetric2
        INTEGER,
    ipRouteMetric3
        INTEGER,
    ipRouteMetric4
        INTEGER,
    ipRouteNextHop
        IpAddress,
    ipRouteType
        INTEGER,
    ipRouteProto
        INTEGER,
    ipRouteAge
        INTEGER
}

ipRouteDest OBJECT-TYPE
        SYNTAX   IpAddress
        ACCESS   read-write
        STATUS   mandatory
        ::= { ipRouteEntry 1 }

ipRouteIfIndex   OBJECT-TYPE
        SYNTAX   INTEGER
        ACCESS   read-write
        STATUS   mandatory
        ::= { ipRouteEntry 2 }
ipRouteMetric1 OBJECT-TYPE
        SYNTAX   INTEGER
```

```
        ACCESS   read-write
        STATUS   mandatory
        ::= { ipRouteEntry 3 }

ipRouteMetric2 OBJECT-TYPE
        SYNTAX   INTEGER
        ACCESS   read-write
        STATUS   mandatory
        ::= { ipRouteEntry 4 }

ipRouteMetric3 OBJECT-TYPE
        SYNTAX   INTEGER
        ACCESS   read-write
        STATUS   mandatory
        ::= { ipRouteEntry 5 }

ipRouteMetric4 OBJECT-TYPE
        SYNTAX   INTEGER
        ACCESS   read-write
        STATUS   mandatory
        ::= { ipRouteEntry 6 }

ipRouteNextHop OBJECT-TYPE
        SYNTAX   IpAddress
        ACCESS   read-write
        STATUS   mandatory
        ::= { ipRouteEntry 7 }

ipRouteType OBJECT-TYPE
        SYNTAX   INTEGER {
          other(1),         -- none of the following

          invalid(2),       -- an invalidated route

                            -- route to directly
          direct(3),        -- connected (sub-)network

                            -- route to a non-local
          remote(4),        -- host/network/sub-network
            }
        ACCESS   read-write
        STATUS   mandatory
        ::= { ipRouteEntry 8 }

ipRouteProto OBJECT-TYPE
        SYNTAX   INTEGER {
          other(1),         -- none of the following

                            -- non-protocol information
                            --     e.g., manually
          local(2),         --     configured entries

                            -- set via a network
          netmgmt(3),       --     management protocol
```

```
                    -- obtained via ICMP,
            icmp(4),        --    e.g., Redirect

                    -- the following are
                    -- gateway routing protocols
            egp(5),
            ggp(6),
            hello(7),
            rip(8),
            is-is(9),
            es-is(10),
            ciscoIgrp(11),
            bbnSpfIgp(12),
            oigp(13)
              }
        ACCESS   read-only
        STATUS   mandatory
        ::= { ipRouteEntry 9 }

ipRouteAge OBJECT-TYPE
        SYNTAX   INTEGER
        ACCESS   read-write
        STATUS   mandatory
        ::= { ipRouteEntry 10 }
```

The ICMP Group

```
icmpInMsgs OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { icmp 1 }

icmpInErrors OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { icmp 2 }

icmpInDestUnreachs OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { icmp 3 }

icmpInTimeExcds OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { icmp 4 }

icmpInParmProbs OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { icmp 5 }

icmpInSrcQuenchs OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { icmp 6 }

icmpInRedirects OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { icmp 7 }

icmpInEchos OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { icmp 8 }

icmpInEchoReps OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
```

```
        STATUS   mandatory
        ::= { icmp 9 }

icmpInTimestamps OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { icmp 10 }

icmpInTimestampReps OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { icmp 11 }

icmpInAddrMasks OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { icmp 12 }

icmpInAddrMaskReps OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { icmp 13 }

icmpOutMsgs OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { icmp 14 }

icmpOutErrors OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { icmp 15 }

icmpOutDestUnreachs OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { icmp 16 }

icmpOutTimeExcds OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { icmp 17 }

icmpOutParmProbs OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
```

```
        ::= { icmp 18 }

icmpOutSrcQuenchs OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { icmp 19 }

icmpOutRedirects OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { icmp 20 }

icmpOutEchos OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { icmp 21 }

icmpOutEchoReps OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { icmp 22 }

icmpOutTimestamps OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { icmp 23 }

icmpOutTimestampReps OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { icmp 24 }

icmpOutAddrMasks OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { icmp 25 }

icmpOutAddrMaskReps OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { icmp 26 }
```

The TCP Group

```
tcpRtoAlgorithm OBJECT-TYPE
        SYNTAX   INTEGER {
        other(1),      -- none of the following
        constant(2), -- a constant rto
        rsre(3),       -- MIL-STD-1778, Appendix B
        vanj(4)        -- Van Jacobson's algorithm [15]
                 }
        ACCESS   read-only
        STATUS   mandatory
        ::= { tcp 1 }

tcpRtoMin OBJECT-TYPE
        SYNTAX   INTEGER
        ACCESS   read-only
        STATUS   mandatory
        ::= { tcp 2 }

tcpRtoMax OBJECT-TYPE
        SYNTAX   INTEGER
        ACCESS   read-only
        STATUS   mandatory
        ::= { tcp 3 }

tcpMaxConn OBJECT-TYPE
        SYNTAX   INTEGER
        ACCESS   read-only
        STATUS   mandatory
        ::= { tcp 4 }

tcpActiveOpens OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { tcp 5 }

tcpPassiveOpens OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { tcp 6 }

tcpAttemptFails OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { tcp 7 }

tcpEstabResets OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
```

```
        ::= { tcp 8 }

tcpCurrEstab OBJECT-TYPE
        SYNTAX   Gauge
        ACCESS   read-only
        STATUS   mandatory
        ::= { tcp 9 }

tcpInSegs OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { tcp 10 }

tcpOutSegs OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { tcp 11 }

tcpRetransSegs OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { tcp 12 }
```

-- the TCP connections table

```
tcpConnTable OBJECT-TYPE
        SYNTAX   SEQUENCE OF TcpConnEntry
        ACCESS   read-only
        STATUS   mandatory
        ::= { tcp 13 }

tcpConnEntry OBJECT-TYPE
        SYNTAX   TcpConnEntry
        ACCESS   read-only
        STATUS   mandatory
        ::= { tcpConnTable 1 }

TcpConnEntry ::= SEQUENCE {
    tcpConnState
        INTEGER,
    tcpConnLocalAddress
        IpAddress,
    tcpConnLocalPort
        INTEGER (0..65535),
    tcpConnRemAddress
        IpAddress,
    tcpConnRemPort
        INTEGER (0..65535)
}

tcpConnState OBJECT-TYPE
        SYNTAX   INTEGER {
```

```
                        closed(1),
                        listen(2),
                        synSent(3),
                        synReceived(4),
                        established(5),
                        finWait1(6),
                        finWait2(7),
                        closeWait(8),
                        lastAck(9),
                        closing(10),
                        timeWait(11)
                    }
        ACCESS   read-only
        STATUS   mandatory
        ::= { tcpConnEntry 1 }

tcpConnLocalAddress OBJECT-TYPE
        SYNTAX   IpAddress
        ACCESS   read-only
        STATUS   mandatory
        ::= { tcpConnEntry 2 }

tcpConnLocalPort OBJECT-TYPE
        SYNTAX   INTEGER (0..65535)
        ACCESS   read-only
        STATUS   mandatory
        ::= { tcpConnEntry 3 }

tcpConnRemAddress OBJECT-TYPE
        SYNTAX   IpAddress
        ACCESS   read-only
        STATUS   mandatory
        ::= { tcpConnEntry 4 }

tcpConnRemPort OBJECT-TYPE
        SYNTAX   INTEGER (0..65535)
        ACCESS   read-only
        STATUS   mandatory
        ::= { tcpConnEntry 5 }
```

The UDP Group

```
udpInDatagrams OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { udp 1 }

udpNoPorts OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { udp 2 }

udpInErrors OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { udp 3 }

udpOutDatagrams OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { udp 4 }
```

The EGP Group

```
egpInMsgs OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { egp 1 }

egpInErrors OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { egp 2 }

egpOutMsgs OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { egp 3 }

egpOutErrors OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        ::= { egp 4 }
```

-- the EGP Neighbor table

```
egpNeighTable OBJECT-TYPE
        SYNTAX   SEQUENCE OF EgpNeighEntry
        ACCESS   read-only
        STATUS   mandatory
        ::= { egp 5 }

egpNeighEntry OBJECT-TYPE
        SYNTAX   EgpNeighEntry
        ACCESS   read-only
        STATUS   mandatory
        ::= { egpNeighTable 1 }

EgpNeighEntry ::= SEQUENCE {
    egpNeighState
        INTEGER,
    egpNeighAddr
        IpAddress
}

egpNeighState OBJECT-TYPE
        SYNTAX   INTEGER {
                    idle(1),
                    acquisition(2),
                    down(3),
                    up(4),
```

```
                        cease(5)
                    }
            ACCESS   read-only
            STATUS   mandatory
            ::= { egpNeighEntry 1 }

        egpNeighAddr OBJECT-TYPE
            SYNTAX   IpAddress
            ACCESS   read-only
            STATUS   mandatory
            ::= { egpNeighEntry 2 }

END
```

Acknowledgements

The initial draft of this memo was heavily influenced by the the HEMS [9] and SNMP [10] MIBs.

Its final form is the result of the suggestions, the dicussions, and the compromises reached by the members of the IETF MIB working group:

Karl Auerbach, Epilogue Technology
K. Ramesh Babu, Excelan
Lawrence Besaw, Hewlett-Packard
Jeffrey D. Case, University of Tennessee at Knoxville
James R. Davin, Proteon
Mark S. Fedor, NYSERNet
Robb Foster, BBN
Phill Gross, The MITRE Corporation
Bent Torp Jensen, Convergent Technology
Lee Labarre, The MITRE Corporation
Dan Lynch, Advanced Computing Environments
Keith McCloghrie, The Wollongong Group
Dave Mackie, 3Com/Bridge
Craig Partridge, BBN (chair)
Jim Robertson, 3Com/Bridge
Marshall T. Rose, The Wollongong Group
Greg Satz, cisco
Martin Lee Schoffstall, Rensselaer Polytechnic Institute
Lou Steinberg, IBM
Dean Throop, Data General
Unni Warrier, Unisys

Authors' Addresses

Keith McCloghrie
The Wollongong Group
1129 San Antonio Road
Palo Alto, CA 04303

Phone: (415) 962-7160
EMail: sytek!kzm@HPLABS.HP.COM

Marshall T. Rose
PSI, Inc.
PSI California Office
P.O. Box 391776
Mountain View, CA 94039

Phone: (415) 961-3380
EMail: mrose@PSI.COM

8. References

[1] Cerf, V., "IAB Recommendations for the Development of Internet Network Management Standards", RFC 1052, IAB, April 1988.

[2] Information processing systems - Open Systems Interconnection, "Management Information Services Definition", International Organization for Standardization, Draft Proposal 9595/2, December 1987.

[3] Information processing systems - Open Systems Interconnection, "Management Information Protocol Specification", International Organization for Standardization, Draft Proposal 9596/2, December 1987.

[4] Rose M., and K. McCloghrie, "Structure and Identification of Management Information for TCP/IP-based internets", RFC 1065, TWG, August 1988.

[5] Partridge C., and G. Trewitt, "The High-Level Entity Management System (HEMS)", RFCs 1021-1024, BBN and Stanford, October 1987.

[6] Cerf, V., "Report of the Second Ad Hoc Network Management Review Group", RFC 1109, IAB, August 1989.

[7] Rose, M., and K. McCloghrie, "Structure and Identification of Management Information for TCP/IP-based Internets", RFC 1155, Performance Systems International and Hughes LAN Systems, May 1990.

[8] Case, J., M. Fedor, M. Schoffstall, and J. Davin, The Simple Network Management Protocol", RFC 1157, University of Tennessee at Knoxville, Performance Systems International, Performance Systems International, and the MIT Laboratory for Computer Science, May 1990.

[9] Partridge C., and G. Trewitt, "HEMS Variable Definitions", RFC

1024, BBN and Stanford, October 1987.

[10] Case, J., M. Fedor, M. Schoffstall, and J. Davin, "A Simple Network Management Protocol", RFC 1067, University of Tennessee At Knoxville, NYSERNet, Rensselaer Polytechnic, Proteon, August 1988.

[11] LaBarre, L., "Structure and Identification of Management Information for the Internet", Internet Engineering Task Force working note, Network Information Center, SRI International, Menlo Park, California, April 1988.

[12] LaBarre, L., "Transport Layer Management Information: TCP", Internet Engineering Task Force working note in preparation. Network Information Center, SRI International, Menlo Park, California, (unpublished).

[13] Information processing systems - Open Systems Interconnection, "Specification of Abstract Syntax Notation One (ASN.1)", International Organization for Standardization, International Standard 8824, December 1987.

[14] Information processing systems - Open Systems Interconnection, "Specification of Basic Encoding Rules for Abstract Notation One (ASN.1)", International Organization for Standardization, International Standard 8825, December 1987.

[15] Jacobson, V., "Congestion Avoidance and Control", SIGCOMM, 1988, Stanford, California.

Security Considerations

Security issues are not discussed in this memo.

RFC-1157 A Simple Network Management Protocol

J. Case, M. Fedor, and J. Davin

May 1990

RFC-1157 A Simple Network Management Protocol

Status of this Memo

This RFC is a re-release of RFC-1098, with a changed "Status of this Memo" section plus a few minor typographical corrections. This memo defines a simple protocol by which management information for a network element may be inspected or altered by logically remote users. In particular, together with its companion memos which describe the structure of management information along with the management information base, these documents provide a simple, workable architecture and system for managing TCP/IP-based internets and in particular the Internet.

The Internet Activities Board recommends that all IP and TCP implementations be network manageable. This implies implementation of the Internet MIB (RFC-1156) and at least one of the two recommended management protocols SNMP (RFC-1157) or CMOT (RFC-1189). It should be noted that, at this time, SNMP is a full Internet standard and CMOT is a draft standard. See also the Host and Gateway Requirements RFCs for more specific information on the applicability of this standard.

Please refer to the latest edition of the "IAB Official Protocol Standards" RFC for current information on the state and status of standard Internet protocols.

Distribution of this memo is unlimited.

RFC-1157 A Simple Network Management Protocol

Introduction

As reported in RFC-1052, IAB Recommendations for the Development of Internet Network Management Standards [1], a two-prong strategy for network management of TCP/IP-based internets was undertaken. In the short-term, the Simple Network Management Protocol (SNMP) was to be used to manage nodes in the Internet community. In the long-term, the use of the OSI network management framework was to be examined. Two documents were produced to define the management information: RFC-1155, which defined the Structure of Management Information (SMI), and RFC-1156, which defined the Management Information Base (MIB). Both of these documents were designed so as to be compatible with both the SNMP and the OSI network management framework.

This strategy was quite successful in the short-term: Internet-based network management technology was fielded, by both the research and commercial communities, within a few months. As a result of this, portions of the Internet community became network manageable in a timely fashion.

As reported in RFC-1109, Report of the Second Ad Hoc Network Management Review Group [4], the requirements of the SNMP and the OSI network management frameworks were more different than anticipated. As such, the requirement for compatibility between the SMI/MIB and both frameworks was suspended. This action permitted the operational network management framework, the SNMP, to respond to new operational needs in the Internet community by producing documents defining new MIB items.

The IAB has designated the SNMP, SMI, and the initial Internet MIB to be full "Standard Protocols" with "Recommended" status. By this action, the IAB recommends that all IP and TCP implementations be network manageable and that the implementations that are network manageable are expected to adopt and implement the SMI, MIB, and SNMP.

As such, the current network management framework for TCP/IP- based internets consists of: Structure and Identification of Management Information for TCP/IP-based Internets, which describes how managed objects contained in the MIB are defined as set forth in RFC-1155; Management Information Base for Network Management of TCP/IP- based Internets, which describes the managed objects contained in the MIB as set forth in RFC-1156; and, the Simple Network Management Protocol, which defines the protocol used to manage these objects, as set forth in this memo.

As reported in RFC-1052, IAB Recommendations for the Development of Internet Network Management Standards [1], the Internet Activities Board has directed the Internet Engineering Task Force (IETF) to create two new working groups in the area of network management. One group was charged with the further specification and definition of elements to be included in the Management Information Base (MIB). The other was charged with defining the modifications to the Simple Network Management Protocol (SNMP) to accommodate the short-term needs of the network vendor and operations communities, and to align with the output of the MIB working group.

The MIB working group produced two memos, one which defines a Structure for Management Information (SMI) [RFC-1155] for use by the managed objects contained in the MIB. A second memo Management Information Base [RFC-1156] defines the list of managed objects.

The output of the SNMP Extensions working group is this memo, which incorporates changes to the initial SNMP definition [7] required to attain alignment with the output of the MIB working group. The changes should be minimal in order to be consistent with the IAB's directive that the working groups be "extremely sensitive to the need to keep the SNMP

simple." Although considerable care and debate has gone into the changes to the SNMP which are reflected in this memo, the resulting protocol is not backwardly-compatible with its predecessor, the Simple Gateway Monitoring Protocol (SGMP) [8]. Although the syntax of the protocol has been altered, the original philosophy, design decisions, and architecture remain intact. In order to avoid confusion, new UDP ports have been allocated for use by the protocol described in this memo.

RFC-1157 A Simple Network Management Protocol

The SNMP Architecture

Implicit in the SNMP architectural model is a collection of network management stations and network elements. Network management stations execute management applications which monitor and control network elements. Network elements are devices such as hosts, gateways, terminal servers, and the like, which have management agents responsible for performing the network management functions requested by the network management stations. The Simple Network Management Protocol (SNMP) is used to communicate management information between the network management stations and the agents in the network elements.

Goals of the Architecture

The SNMP explicitly minimizes the number and complexity of management functions realized by the management agent itself. This goal is attractive in at least four respects:

(1) The development cost for management agent software necessary to support the protocol is accordingly reduced.

(2) The degree of management function that is remotely supported is accordingly increased, thereby admitting fullest use of internet resources in the management task.

(3) The degree of management function that is remotely supported is accordingly increased, thereby imposing the fewest possible restrictions on the form and sophistication of management tools.

(4) Simplified sets of management functions are easily understood and used by developers of network management tools.

A second goal of the protocol is that the functional paradigm for monitoring and control be sufficiently extensible to accommodate additional, possibly unanticipated aspects of network operation and management.

A third goal is that the architecture be, as much as possible, independent of the architecture and mechanisms of particular hosts or particular gateways.

Elements of the Architecture

The SNMP architecture articulates a solution to the network management problem in terms of:

(1) the scope of the management information communicated by the protocol,

(2) the representation of the management information communicated by the protocol,

(3) operations on management information supported by the protocol,

(4) the form and meaning of exchanges among management entities,

(5) the definition of administrative relationships among management entities, and

(6) the form and meaning of references to management information.

Scope of Management Information

The scope of the management information communicated by operation of the SNMP is exactly that represented by instances of all non- aggregate object types either defined in Internet-standard MIB or defined elsewhere according to the conventions set forth in Internet-standard SMI [RFC-1155].

Support for aggregate object types in the MIB is neither required for conformance with the SMI nor realized by the SNMP.

Representation of Management Information

Management information communicated by operation of the SNMP is represented according to the subset of the ASN.1 language [9] that is specified for the definition of non-aggregate types in the SMI.

The SGMP adopted the convention of using a well-defined subset of the ASN.1 language [9]. The SNMP continues and extends this tradition by utilizing a moderately more complex subset of ASN.1 for describing managed objects and for describing the protocol data units used for managing those objects. In addition, the desire to ease eventual transition to OSI-based network management protocols led to the definition in the ASN.1 language of an Internet-standard Structure of Management Information (SMI) [RFC-1155] and Management Information Base (MIB) [RFC-1156]. The use of the ASN.1 language, was, in part, encouraged by the successful use of ASN.1 in earlier efforts, in particular, the SGMP. The restrictions on the use of ASN.1 that are part of the SMI contribute to the simplicity espoused and validated by experience with the SGMP.

Also for the sake of simplicity, the SNMP uses only a subset of the basic encoding rules of ASN.1 [10]. Namely, all encodings use the definite-length form. Further, whenever permissible, non-constructor encodings are used rather than constructor encodings. This restriction applies to all aspects of ASN.1 encoding, both for the top-level protocol data units and the data objects they contain.

Operations Supported on Management Information

The SNMP models all management agent functions as alterations or inspections of variables. Thus, a protocol entity on a logically remote host (possibly the network element itself) interacts with the management agent resident on the network element in order to retrieve (get) or alter (set) variables. This strategy has at least two positive consequences:

(1) It has the effect of limiting the number of essential management functions realized by the management agent to two: one operation to assign a value to a specified configuration or other parameter and another to retrieve such a value.

(2) A second effect of this decision is to avoid introducing into the protocol definition support for imperative management commands: the number of such commands is in practice ever-increasing, and the semantics of such commands are in general arbitrarily complex.

The strategy implicit in the SNMP is that the monitoring of network state at any significant level of detail is accomplished primarily by polling for appropriate information on the part of the monitoring center(s). A limited number of unsolicited messages (traps) guide the timing and focus of the polling. Limiting the number of unsolicited messages is consistent with the goal of simplicity and minimizing the amount of traffic generated by the network management function.

The exclusion of imperative commands from the set of explicitly supported management functions is unlikely to preclude any desirable management agent operation. Currently, most commands are requests either to set the value of some parameter or to retrieve such a value, and the function of the few imperative commands currently supported is easily accommodated in an asynchronous mode by this management model. In this scheme, an imperative command might be realized as the setting of a parameter value that subsequently triggers the desired action. For example, rather than implementing a "reboot command," this action might be invoked by simply setting a parameter indicating the number of seconds until system reboot.

RFC-1157 SNMP - Elements of the Architecture

Form and Meaning of Protocol Exchanges

The communication of management information among management entities is realized in the SNMP through the exchange of protocol messages.

Consistent with the goal of minimizing complexity of the management agent, the exchange of SNMP messages requires only an unreliable datagram service, and every message is entirely and independently represented by a single transport datagram. While this document specifies the exchange of messages via the UDP protocol, the mechanisms of the SNMP are generally suitable for use with a wide variety of transport services.

Definition of Administrative Relationships

The SNMP architecture admits a variety of administrative relationships among entities that participate in the protocol. The entities residing at management stations and network elements which communicate with one another using the SNMP are termed SNMP application entities. The peer processes which implement the SNMP, and thus support the SNMP application entities, are termed protocol entities.

A pairing of an SNMP agent with some arbitrary set of SNMP application entities is called an SNMP community. Each SNMP community is named by a string of octets, that is called the community name for said community.

An SNMP message originated by an SNMP application entity that in fact belongs to the SNMP community named by the community component of said message is called an authentic SNMP message. The set of rules by which an SNMP message is identified as an authentic SNMP message for a particular SNMP community is called an authentication scheme. An implementation of a function that identifies authentic SNMP messages according to one or more authentication schemes is called an authentication service.

Clearly, effective management of administrative relationships among SNMP application entities requires authentication services that (by the use of encryption or other techniques) are able to identify authentic SNMP messages with a high degree of certainty. Some SNMP implementations may wish to support only a trivial authentication service that identifies all SNMP messages as authentic SNMP messages.

For any network element, a subset of objects in the MIB that pertain to that element is called a SNMP MIB view. Note that the names of the object types represented in a SNMP MIB view need not belong to a single sub-tree of the object type name space.

An element of the set { READ-ONLY, READ-WRITE } is called an SNMP access mode.

A pairing of a SNMP access mode with a SNMP MIB view is called an SNMP community profile. A SNMP community profile represents specified access privileges to variables in a specified MIB view. For every variable in the MIB view in a given SNMP community profile, access to that variable is represented by the profile according to the following conventions:

(1) if said variable is defined in the MIB with "Access:" of "none," it is unavailable as an operand for any operator;

(2) if said variable is defined in the MIB with "Access:" of "read-write" or "write-only" and the access mode of the given profile is READ-WRITE, that variable is available as an operand for the get, set, and trap operations;

(3) otherwise, the variable is available as an operand for the get and trap operations.

(4) In those cases where a "write-only" variable is an operand used for the get or trap operations, the value given for the variable is implementation-specific.

A pairing of a SNMP community with a SNMP community profile is called a SNMP access policy. An access policy represents a specified community profile afforded by the SNMP agent of a specified SNMP community to other members of that community. All administrative relationships among SNMP application entities are architecturally defined in terms of SNMP access policies.

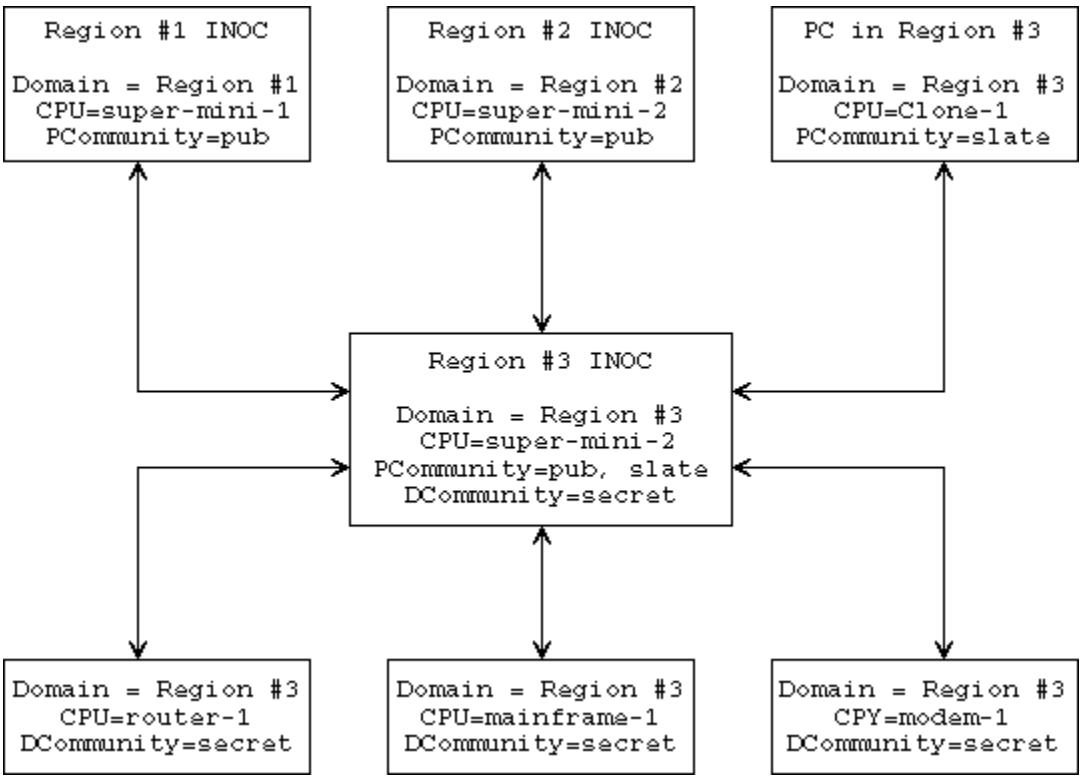
For every SNMP access policy, if the network element on which the SNMP agent for the

specified SNMP community resides is not that to which the MIB view for the specified profile pertains, then that policy is called a SNMP proxy access policy. The SNMP agent associated with a proxy access policy is called a SNMP proxy agent. While careless definition of proxy access policies can result in management loops, prudent definition of proxy policies is useful in at least two ways:

(1) It permits the monitoring and control of network elements which are otherwise not addressable using the management protocol and the transport protocol. That is, a proxy agent may provide a protocol conversion function allowing a management station to apply a consistent management framework to all network elements, including devices such as modems, multiplexors, and other devices which support different management frameworks.

(2) It potentially shields network elements from elaborate access control policies. For example, a proxy agent may implement sophisticated access control whereby diverse subsets of variables within the MIB are made accessible to different management stations without increasing the complexity of the network element.

By way of example, Figure 1 illustrates the relationship between management stations, proxy agents, and management agents. In this example, the proxy agent is envisioned to be a normal Internet Network Operations Center (INOC) of some administrative domain which has a standard managerial relationship with a set of management agents.



Domain: The administrative domain of the element
PCommunity: the name of a community utilizing a proxy agent
DCommunity: the name of a direct community

Form and Meaning of References to Managed Objects

The SMI requires that the definition of a conformant management protocol address:

(1) the resolution of ambiguous MIB references,

(2) the resolution of MIB references in the presence multiple MIB versions, and

(3) the identification of particular instances of object types defined in the MIB.

Resolution of Ambiguous MIB References

Because the scope of any SNMP operation is conceptually confined to objects relevant to a single network element, and because all SNMP references to MIB objects are (implicitly or explicitly) by unique variable names, there is no possibility that any SNMP reference to any object type defined in the MIB could resolve to multiple instances of that type.

Resolution of References across MIB Versions

The object instance referred to by any SNMP operation is exactly that specified as part of the operation request or (in the case of a get- next operation) its immediate successor in the MIB as a whole. In particular, a reference to an object as part of some version of the Internet-standard MIB does not resolve to any object that is not part of said version of the Internet-standard MIB, except in the case that the requested operation is get-next and the specified object name is lexicographically last among the names of all objects presented as part of said version of the Internet-Standard MIB.

RFC-1157 SNMP - Form and Meaning of References to Managed Objects

Identification of Object Instances

The names for all object types in the MIB are defined explicitly either in the Internet-standard MIB or in other documents which conform to the naming conventions of the SMI. The SMI requires that conformant management protocols define mechanisms for identifying individual instances of those object types for a particular network element.

Each instance of any object type defined in the MIB is identified in SNMP operations by a unique name called its "variable name." In general, the name of an SNMP variable is an OBJECT IDENTIFIER of the form x.y, where x is the name of a non-aggregate object type defined in the MIB and y is an OBJECT IDENTIFIER fragment that, in a way specific to the named object type, identifies the desired instance.

This naming strategy admits the fullest exploitation of the semantics of the GetNextRequest-PDU, because it assigns names for related variables so as to be contiguous in the lexicographical ordering of all variable names known in the MIB.

The type-specific naming of object instances is defined below for a number of classes of object types. Instances of an object type to which none of the following naming conventions are applicable are named by OBJECT IDENTIFIERs of the form x.0, where x is the name of said object type in the MIB definition.

For example, suppose one wanted to identify an instance of the variable sysDescr The object class for sysDescr is:

iso	org	dod	internet	mgmt	mib	system	sysDescr
1	3	6	1	2	1	1	1

Hence, the object type, x, would be 1.3.6.1.2.1.1.1 to which is appended an instance sub-identifier of 0. That is, 1.3.6.1.2.1.1.1.0 identifies the one and only instance of sysDescr.

ifTable Object Type Names
atTable Object Type Names
ipAddrTable Object Type Names
ipRoutingTable Object Type Names
tcpConnTable Object Type Names
egpNeighTable Object Type Names

ifTable Object Type Names

The name of a subnet interface, s, is the OBJECT IDENTIFIER value of the form i, where i has the value of that instance of the ifIndex object type associated with s.

For each object type, t, for which the defined name, n, has a prefix of ifEntry, an instance, i, of t is named by an OBJECT IDENTIFIER of the form n.s, where s is the name of the subnet interface about which i represents information.

For example, suppose one wanted to identify the instance of the variable ifType associated with interface 2. Accordingly, ifType.2 would identify the desired instance.

atTable Object Type Names

The name of an AT-cached network address, x, is an OBJECT IDENTIFIER of the form 1.a.b.c.d, where a.b.c.d is the value (in the familiar "dot" notation) of the atNetAddress object type associated with x.

The name of an address translation equivalence e is an OBJECT IDENTIFIER value of the form s.w, such that s is the value of that instance of the atIndex object type associated with e and such that w is the name of the AT-cached network address associated with e.

For each object type, t, for which the defined name, n, has a prefix of atEntry, an instance, i, of t is named by an OBJECT IDENTIFIER of the form n.y, where y is the name of the address translation equivalence about which i represents information.

For example, suppose one wanted to find the physical address of an entry in the address translation table (ARP cache) associated with an IP address of 89.1.1.42 and interface 3. Accordingly, atPhysAddress.3.1.89.1.1.42 would identify the desired instance.

ipAddrTable Object Type Names

The name of an IP-addressable network element, x, is the OBJECT IDENTIFIER of the form a.b.c.d such that a.b.c.d is the value (in the familiar "dot" notation) of that instance of the ipAdEntAddr object type associated with x.

For each object type, t, for which the defined name, n, has a prefix of ipAddrEntry, an instance, i, of t is named by an OBJECT IDENTIFIER of the form n.y, where y is the name of the IP-addressable network element about which i represents information.

For example, suppose one wanted to find the network mask of an entry in the IP interface table associated with an IP address of 89.1.1.42. Accordingly, ipAdEntNetMask.89.1.1.42 would identify the desired instance.

ipRoutingTable Object Type Names

The name of an IP route, x, is the OBJECT IDENTIFIER of the form a.b.c.d such that a.b.c.d is the value (in the familiar "dot" notation) of that instance of the ipRouteDest object type associated with x.

For each object type, t, for which the defined name, n, has a prefix of ipRoutingEntry, an instance, i, of t is named by an OBJECT IDENTIFIER of the form n.y, where y is the name of the IP route about which i represents information.

For example, suppose one wanted to find the next hop of an entry in the IP routing table associated with the destination of 89.1.1.42. Accordingly, ipRouteNextHop.89.1.1.42 would identify the desired instance.

tcpConnTable Object Type Names

The name of a TCP connection, x, is the OBJECT IDENTIFIER of the form a.b.c.d.e.f.g.h.i.j such that a.b.c.d is the value (in the familiar "dot" notation) of that instance of the tcpConnLocalAddress object type associated with x and such that f.g.h.i is the value (in the familiar "dot" notation) of that instance of the tcpConnRemoteAddress object type associated with x and such that e is the value of that instance of the tcpConnLocalPort object type associated with x and such that j is the value of that instance of the tcpConnRemotePort object type associated with x.

For each object type, t, for which the defined name, n, has a prefix of tcpConnEntry, an instance, i, of t is named by an OBJECT IDENTIFIER of the form n.y, where y is the name of the TCP connection about which i represents information.

For example, suppose one wanted to find the state of a TCP connection between the local address of 89.1.1.42 on TCP port 21 and the remote address of 10.0.0.51 on TCP port 2059. Accordingly, tcpConnState.89.1.1.42.21.10.0.0.51.2059 would identify the desired instance.

egpNeighTable Object Type Names

The name of an EGP neighbor, x, is the OBJECT IDENTIFIER of the form a.b.c.d such that a.b.c.d is the value (in the familiar "dot" notation) of that instance of the egpNeighAddr object type associated with x.

For each object type, t, for which the defined name, n, has a prefix of egpNeighEntry, an instance, i, of t is named by an OBJECT IDENTIFIER of the form n.y, where y is the name of the EGP neighbor about which i represents information.

For example, suppose one wanted to find the neighbor state for the IP address of 89.1.1.42. Accordingly, egpNeighState.89.1.1.42 would identify the desired instance.

RFC-1157 A Simple Network Management Protocol

Protocol Specification

The network management protocol is an application protocol by which the variables of an agent's MIB may be inspected or altered.

Communication among protocol entities is accomplished by the exchange of messages, each of which is entirely and independently represented within a single UDP datagram using the basic encoding rules of ASN.1 (as discussed in Section 3.2.2). A message consists of a version identifier, an SNMP community name, and a protocol data unit (PDU). A protocol entity receives messages at UDP port 161 on the host with which it is associated for all messages except for those which report traps (i.e., all messages except those which contain the Trap-PDU). Messages which report traps should be received on UDP port 162 for further processing. An implementation of this protocol need not accept messages whose length exceeds 484 octets. However, it is recommended that implementations support larger datagrams whenever feasible.

It is mandatory that all implementations of the SNMP support the five PDUs: GetRequest-PDU, GetNextRequest-PDU, GetResponse-PDU, SetRequest-PDU, and Trap-PDU.

```
RFC1157-SNMP DEFINITIONS ::= BEGIN
    IMPORTS
        ObjectName, ObjectSyntax, NetworkAddress, IpAddress, TimeTicks
    FROM RFC1155-SMI;
-- top-level message
Message ::=
    SEQUENCE {
        version          -- version-1 for this RFC
        INTEGER {
            version-1(0)
        },
        community        -- community name
        OCTET STRING,
            data             -- e.g., PDUs if trivial
            ANY          -- authentication is being used
        }
-- protocol data units
PDUs ::=
    CHOICE {
        get-request
            GetRequest-PDU,
        get-next-request
             GetNextRequest-PDU,
        get-response
            GetResponse-PDU,
        set-request
            SetRequest-PDU,
        trap
            Trap-PDU
    }
-- the individual PDUs and commonly used
-- data types will be defined later
END
```

Elements of Procedure

This section describes the actions of a protocol entity implementing the SNMP. Note, however, that it is not intended to constrain the internal architecture of any conformant implementation.

In the text that follows, the term transport address is used. In the case of the UDP, a transport address consists of an IP address along with a UDP port. Other transport services may be used to support the SNMP. In these cases, the definition of a transport address should be made accordingly.

The top-level actions of a protocol entity which generates a message are as follows:

(1) It first constructs the appropriate PDU, e.g., the GetRequest-PDU, as an ASN.1 object.

(2) It then passes this ASN.1 object along with a community name its source transport address and the destination transport address, to the service which implements the desired authentication scheme. This authentication service returns another ASN.1 object.

(3) The protocol entity then constructs an ASN.1 Message object, using the community name and the resulting ASN.1 object.

(4) This new ASN.1 object is then serialized, using the basic encoding rules of ASN.1, and then sent using a transport service to the peer protocol entity.

Similarly, the top-level actions of a protocol entity which receives a message are as follows:

(1) It performs a rudimentary parse of the incoming datagram to build an ASN.1 object corresponding to an ASN.1 Message object. If the parse fails, it discards the datagram and performs no further actions.

(2) It then verifies the version number of the SNMP message. If there is a mismatch, it discards the datagram and performs no further actions.

(3) The protocol entity then passes the community name and user data found in the ASN.1 Message object, along with the datagram's source and destination transport addresses to the service which implements the desired authentication scheme. This entity returns another ASN.1 object, or signals an authentication failure. In the latter case, the protocol entity notes this failure, (possibly) generates a trap, and discards the datagram and performs no further actions.

(4) The protocol entity then performs a rudimentary parse on the ASN.1 object returned from the authentication service to build an ASN.1 object corresponding to an ASN.1 PDUs object. If the parse fails, it discards the datagram and performs no further actions. Otherwise, using the named SNMP community, the appropriate profile is selected, and the PDU is processed accordingly. If, as a result of this processing, a message is returned then the source transport address that the response message is sent from shall be identical to the destination transport address that the original request message was sent to.

Common Constructs
The GetRequest-PDU
The GetNextRequest-PDU

RFC-1157 SNMP - Elements of Procdedure

Common Constructs

Before introducing the six PDU types of the protocol, it is appropriate to consider some of the ASN.1 constructs used frequently:

```
-- request/response information
RequestID ::=
    INTEGER
ErrorStatus ::=
    INTEGER {
        noError(0),
        tooBig(1),
        noSuchName(2),
        badValue(3),
        readOnly(4)
        genErr(5)
    }
ErrorIndex ::=
    INTEGER
-- variable bindings
VarBind ::=
    SEQUENCE {
        name
            ObjectName,
        value
            ObjectSyntax
    }
VarBindList ::=
    SEQUENCE OF
        VarBind
```

RequestIDs are used to distinguish among outstanding requests. By use of the RequestID, an SNMP application entity can correlate incoming responses with outstanding requests. In cases where an unreliable datagram service is being used, the RequestID also provides a simple means of identifying messages duplicated by the network.

A non-zero instance of ErrorStatus is used to indicate that an exception occurred while processing a request. In these cases, ErrorIndex may provide additional information by indicating which variable in a list caused the exception.

The term variable refers to an instance of a managed object. A variable binding, or VarBind, refers to the pairing of the name of a variable to the variable's value. A VarBindList is a simple list of variable names and corresponding values. Some PDUs are concerned only with the name of a variable and not its value (e.g., the GetRequest-PDU). In this case, the value portion of the binding is ignored by the protocol entity. However, the value portion must still have valid ASN.1 syntax and encoding. It is recommended that the ASN.1 value NULL be used for the value portion of such bindings.

The GetRequest-PDU

The form of the GetRequest-PDU is:

```
GetRequest-PDU ::=
    [0]
    IMPLICIT SEQUENCE {
        request-id
            RequestID,
        error-status            -- always 0
            ErrorStatus,
        error-index             -- always 0
            ErrorIndex,
        variable-bindings
                VarBindList
    }
```

The GetRequest-PDU is generated by a protocol entity only at the request of its SNMP application entity.

Upon receipt of the GetRequest-PDU, the receiving protocol entity responds according to any applicable rule in the list below:

(1) If, for any object named in the variable-bindings field, the object's name does not exactly match the name of some object available for get operations in the relevant MIB view, then the receiving entity sends to the originator of the received message the GetResponse-PDU of identical form, except that the value of the error-status field is noSuchName, and the value of the error-index field is the index of said object name component in the received message.

(2) If, for any object named in the variable-bindings field, the object is an aggregate type (as defined in the SMI), then the receiving entity sends to the originator of the received message the GetResponse-PDU of identical form, except that the value of the error-status field is noSuchName, and the value of the error-index field is the index of said object name component in the received message.

(3) If the size of the GetResponse-PDU generated as described below would exceed a local limitation, then the receiving entity sends to the originator of the received message the GetResponse-PDU of identical form, except that the value of the error-status field is tooBig, and the value of the error-index field is zero.

(4) If, for any object named in the variable-bindings field, the value of the object cannot be retrieved for reasons not covered by any of the foregoing rules, then the receiving entity sends to the originator of the received message the GetResponse-PDU of identical form, except that the value of the error-status field is genErr and the value of the error-index field is the index of said object name component in the received message.

If none of the foregoing rules apply, then the receiving protocol entity sends to the originator of the received message the GetResponse-PDU such that, for each object named in the variable- bindings field of the received message, the corresponding component of the GetResponse-PDU represents the name and value of that variable. The value of the error-

status field of the GetResponse- PDU is noError and the value of the error-index field is zero. The value of the request-id field of the GetResponse-PDU is that of the received message.

The GetNextRequest-PDU

The form of the GetNextRequest-PDU is identical to that of the GetRequest-PDU except for the indication of the PDU type. In the ASN.1 language:

```
GetNextRequest-PDU ::=
[1]
    IMPLICIT SEQUENCE {
        request-id
            RequestID,
        error-status            -- always 0
            ErrorStatus,
        error-index             -- always 0
            ErrorIndex,
        variable-bindings
            VarBindList
    }
```

The GetNextRequest-PDU is generated by a protocol entity only at the request of its SNMP application entity.

Upon receipt of the GetNextRequest-PDU, the receiving protocol entity responds according to any applicable rule in the list below:

(1) If, for any object name in the variable-bindings field, that name does not lexicographically precede the name of some object available for get operations in the relevant MIB view, then the receiving entity sends to the originator of the received message the GetResponse-PDU of identical form, except that the value of the error-status field is noSuchName, and the value of the error-index field is the index of said object name component in the received message.

(2) If the size of the GetResponse-PDU generated as described below would exceed a local limitation, then the receiving entity sends to the originator of the received message the GetResponse-PDU of identical form, except that the value of the error-status field is tooBig, and the value of the error-index field is zero.

(3) If, for any object named in the variable-bindings field, the value of the lexicographical successor to the named object cannot be retrieved for reasons not covered by any of the foregoing rules, then the receiving entity sends to the originator of the received message the GetResponse-PDU of identical form, except that the value of the error-status field is genErr and the value of the error-index field is the index of said object name component in the received message.

If none of the foregoing rules apply, then the receiving protocol entity sends to the originator of the received message the GetResponse-PDU such that, for each name in the variable-bindings field of the received message, the corresponding component of the GetResponse-PDU represents the name and value of that object whose name is, in the lexicographical ordering of the names of all objects available for get operations in the relevant MIB view, together with the value of the name field of the given component, the immediate successor to that value. The value of the error-status field of the GetResponse-PDU is noError and the value of the errorindex field is zero. The value of the request-id field of the GetResponse-

PDU is that of the received message.

RFC-1157 SNMP - Elements of Procedure

Example of Table Traversal

One important use of the GetNextRequest-PDU is the traversal of conceptual tables of information within the MIB. The semantics of this type of SNMP message, together with the protocol-specific mechanisms for identifying individual instances of object types in the MIB, affords access to related objects in the MIB as if they enjoyed a tabular organization.

By the SNMP exchange sketched below, an SNMP application entity might extract the destination address and next hop gateway for each entry in the routing table of a particular network element. Suppose that this routing table has three entries:

Destination	NextHop	Metric
10.0.0.99	89.1.1.42	5
9.1.2.3	99.0.0.3	3
10.0.0.51	89.1.1.42	5

The management station sends to the SNMP agent a GetNextRequest-PDU containing the indicated OBJECT IDENTIFIER values as the requested variable names:

```
GetNextRequest ( ipRouteDest, ipRouteNextHop, ipRouteMetric1 )
```

The SNMP agent responds with a GetResponse-PDU:

```
GetResponse (( ipRouteDest.9.1.2.3 =   "9.1.2.3" ),
    ( ipRouteNextHop.9.1.2.3 = "99.0.0.3" ),
    ( ipRouteMetric1.9.1.2.3 = 3 ))
```

The management station continues with:

```
GetNextRequest ( ipRouteDest.9.1.2.3,
    ipRouteNextHop.9.1.2.3,
    ipRouteMetric1.9.1.2.3 )
```

The SNMP agent responds:

```
GetResponse (( ipRouteDest.10.0.0.51 = "10.0.0.51" ),
    ( ipRouteNextHop.10.0.0.51 = "89.1.1.42" ),
    ( ipRouteMetric1.10.0.0.51 = 5 ))
```

The management station continues with:

```
GetNextRequest ( ipRouteDest.10.0.0.51,
    ipRouteNextHop.10.0.0.51,
    ipRouteMetric1.10.0.0.51 )
```

The SNMP agent responds:

```
GetResponse (( ipRouteDest.10.0.0.99 = "10.0.0.99" ),
    ( ipRouteNextHop.10.0.0.99 = "89.1.1.42" ),
    ( ipRouteMetric1.10.0.0.99 = 5 ))
```

The management station continues with:

```
GetNextRequest ( ipRouteDest.10.0.0.99,
    ipRouteNextHop.10.0.0.99,
    ipRouteMetric1.10.0.0.99 )
```

As there are no further entries in the table, the SNMP agent returns those objects that are next in the lexicographical ordering of the known object names. This response signals the end of the routing table to the management station.

The GetResponse-PDU

The form of the GetResponse-PDU is identical to that of the GetRequest-PDU except for the indication of the PDU type. In the ASN.1 language:

```
GetResponse-PDU ::=
[2]
    IMPLICIT SEQUENCE {
        request-id
            RequestID,
        error-status
            ErrorStatus,
        error-index
            ErrorIndex,
        variable-bindings
            VarBindList
    }
```

The GetResponse-PDU is generated by a protocol entity only upon receipt of the GetRequest-PDU, GetNextRequest-PDU, or SetRequest-PDU, as described elsewhere in this document.

Upon receipt of the GetResponse-PDU, the receiving protocol entity presents its contents to its SNMP application entity.

The SetRequest-PDU

The form of the SetRequest-PDU is identical to that of the GetRequest-PDU except for the indication of the PDU type. In the ASN.1 language:

```
SetRequest-PDU ::=
 [3]
     IMPLICIT SEQUENCE {
         request-id
             RequestID,
         error-status            -- always 0
             ErrorStatus,
         error-index             -- always 0
             ErrorIndex,
         variable-bindings
             VarBindList
     }
```

The SetRequest-PDU is generated by a protocol entity only at the request of its SNMP application entity.

Upon receipt of the SetRequest-PDU, the receiving entity responds according to any applicable rule in the list below:

(1) If, for any object named in the variable-bindings field, the object is not available for set operations in the relevant MIB view, then the receiving entity sends to the originator of the received message the GetResponse-PDU of identical form, except that the value of the error-status field is noSuchName, and the value of the error-index field is the index of said object name component in the received message.

(2) If, for any object named in the variable-bindings field, the contents of the value field does not, according to the ASN.1 language, manifest a type, length, and value that is consistent with that required for the variable, then the receiving entity sends to the originator of the received message the GetResponse-PDU of identical form, except that the value of the error-status field is badValue, and the value of the error-index field is the index of said object name in the received message.

(3) If the size of the Get Response type message generated as described below would exceed a local limitation, then the receiving entity sends to the originator of the received message the GetResponse-PDU of identical form, except that the value of the error-status field is tooBig, and the value of the error-index field is zero.

(4) If, for any object named in the variable-bindings field, the value of the named object cannot be altered for reasons not covered by any of the foregoing rules, then the receiving entity sends to the originator of the received message the GetResponse-PDU of identical form, except that the value of the error-status field is genErr and the value of the error-index field is the index of said object name component in the received message.

If none of the foregoing rules apply, then for each object named in the variable-bindings field of the received message, the corresponding value is assigned to the variable. Each variable

assignment specified by the SetRequest-PDU should be effected as if simultaneously set with respect to all other assignments specified in the same message.

The receiving entity then sends to the originator of the received message the GetResponse-PDU of identical form except that the value of the error-status field of the generated message is noError and the value of the error-index field is zero.

The Trap-PDU

The form of the Trap-PDU is:

```
Trap-PDU ::=
[4]
    IMPLICIT SEQUENCE {
        enterprise      -- type of object generating
                -- trap, see sysObjectID in [5]
            OBJECT IDENTIFIER,
        agent-addr              -- address of object generating
            NetworkAddress, -- trap
        generic-trap            -- generic trap type
            INTEGER {
                coldStart(0),
                warmStart(1),
                linkDown(2),
                linkUp(3),
                authenticationFailure(4),
                egpNeighborLoss(5),
                enterpriseSpecific(6)
                },
        specific-trap   -- specific code, present even
            INTEGER,    -- if generic-trap is not
                -- enterpriseSpecific
        time-stamp      -- time elapsed between the last
            TimeTicks,  -- (re)initialization of the network
                -- entity and the generation of the
                -- trap
        variable-bindings       -- "interesting" information
            VarBindList
    }
```

The Trap-PDU is generated by a protocol entity only at the request of the SNMP application entity. The means by which an SNMP application entity selects the destination addresses of the SNMP application entities is implementation-specific.

Upon receipt of the Trap-PDU, the receiving protocol entity presents its contents to its SNMP application entity.

The significance of the variable-bindings component of the Trap-PDU is implementation-specific.

Interpretations of the value of the generic-trap field are:

RFC-1157 SNMP - Elements of Procedure: Traps

The coldStart Trap

A coldStart(0) trap signifies that the sending protocol entity is reinitializing itself such that the agent's configuration or the protocol entity implementation may be altered.

The warmStart Trap

A warmStart(1) trap signifies that the sending protocol entity is reinitializing itself such that neither the agent configuration nor the protocol entity implementation is altered.

RFC-1157 SNMP - Elements of Procedure: Traps

The linkDown Trap

A linkDown(2) trap signifies that the sending protocol entity recognizes a failure in one of the communication links represented in the agent's configuration.

The Trap-PDU of type linkDown contains as the first element of its variable-bindings, the name and value of the ifIndex instance for the affected interface.

RFC-1157 SNMP - Elements of Procedure: Traps

The linkUp Trap

A linkUp(3) trap signifies that the sending protocol entity recognizes that one of the communication links represented in the agent's configuration has come up.

The Trap-PDU of type linkUp contains as the first element of its variable-bindings, the name and value of the ifIndex instance for the affected interface.

The authenticationFailure Trap

An authenticationFailure(4) trap signifies that the sending protocol entity is the addressee of a protocol message that is not properly authenticated. While implementations of the SNMP must be capable of generating this trap, they must also be capable of suppressing the emission of such traps via an implementation-specific mechanism.

The egpNeighborLoss Trap

An egpNeighborLoss(5) trap signifies that an EGP neighbor for whom the sending protocol entity was an EGP peer has been marked down and the peer relationship no longer obtains.

The Trap-PDU of type egpNeighborLoss contains as the first element of its variable-bindings, the name and value of the egpNeighAddr instance for the affected neighbor.

The enterpriseSpecific Trap

A enterpriseSpecific(6) trap signifies that the sending protocol entity recognizes that some enterprise-specific event has occurred. The specific-trap field identifies the particular trap which occurred.

RFC-1157 A Simple Network Management Protocol

Definitions

```
RFC1157-SNMP DEFINITIONS ::= BEGIN
IMPORTS
    ObjectName, ObjectSyntax, NetworkAddress, IpAddress, TimeTicks
    FROM RFC1155-SMI;
    -- top-level message
    Message ::=
        SEQUENCE {
            version             -- version-1 for this RFC
                INTEGER {
                    version-1(0)
                },
            community           -- community name
                OCTET STRING,
                    data        -- e.g., PDUs if trivial
                    ANY         -- authentication is being used
        }
    -- protocol data units
    PDUs ::=
        CHOICE {
            get-request
                GetRequest-PDU,
            get-next-request
                GetNextRequest-PDU,
            get-response
                GetResponse-PDU,
            set-request
                SetRequest-PDU,
            trap
                Trap-PDU
                        }
    -- PDUs
        GetRequest-PDU ::=
        [0]
            IMPLICIT PDU
        GetNextRequest-PDU ::=
        [1]
            IMPLICIT PDU
        GetResponse-PDU ::=
        [2]
            IMPLICIT PDU
        SetRequest-PDU ::=
        [3]
            IMPLICIT PDU
        PDU ::=
            SEQUENCE {
                request-id
                    INTEGER,
                 error-status        -- sometimes ignored
                    INTEGER {
                        noError(0),
```

```
                    tooBig(1),
                    noSuchName(2),
                    badValue(3),
                    readOnly(4),
                    genErr(5)
                },
            error-index         -- sometimes ignored
                INTEGER,
            variable-bindings -- values are sometimes ignored
                VarBindList
        }
    Trap-PDU ::=
    [4]
        IMPLICIT SEQUENCE {
            enterprise      -- type of object generating
                            -- trap, see sysObjectID in [5]
            OBJECT IDENTIFIER,
            agent-addr          -- address of object generating
                NetworkAddress, -- trap
            generic-trap        -- generic trap type
                INTEGER {
                    coldStart(0),
                    warmStart(1),
                    linkDown(2),
                    linkUp(3),
                    authenticationFailure(4),
                    egpNeighborLoss(5),
                    enterpriseSpecific(6)
                },
            specific-trap   -- specific code, present even
                INTEGER,  -- if generic-trap is not
                    -- enterpriseSpecific
            time-stamp    -- time elapsed between the last
            TimeTicks,      -- (re)initialization of the network
                    -- entity and the generation of the
                    -- trap
            variable-bindings       -- "interesting" information
                VarBindList
         }

-- variable bindings
    VarBind ::=
        SEQUENCE {
            name
                ObjectName,
            value
                ObjectSyntax
        }
    VarBindList ::=
        SEQUENCE OF
            VarBind
END
```

Acknowledgements

This memo was influenced by the IETF SNMP Extensions working group:

Karl Auerbach, Epilogue Technology
K. Ramesh Babu, Excelan
Amatzia Ben-Artzi, 3Com/Bridge
Lawrence Besaw, Hewlett-Packard
Jeffrey D. Case, University of Tennessee at Knoxville
Anthony Chung, Sytek
James Davidson, The Wollongong Group
James R. Davin, MIT Laboratory for Computer Science
Mark S. Fedor, NYSERNet
Phill Gross, The MITRE Corporation
Satish Joshi, ACC
Dan Lynch, Advanced Computing Environments
Keith McCloghrie, The Wollongong Group
Marshall T. Rose, The Wollongong Group (chair)
Greg Satz, cisco
Martin Lee Schoffstall, Rensselaer Polytechnic Institute
Wengyik Yeong, NYSERNet

Authors' Addresses

Jeffrey D. Case
SNMP Research
P.O. Box 8593
Knoxville, TN 37996-4800

Phone: (615) 573-1434
Email: case@CS.UTK.EDU

Mark Fedor
Performance Systems International
Rensselaer Technology Park
125 Jordan Road
Troy, NY 12180

Phone: (518) 283-8860
Email: fedor@patton.NYSER.NET

Martin Lee Schoffstall
Performance Systems International
Rensselaer Technology Park
165 Jordan Road
Troy, NY 12180

Phone: (518) 283-8860
Email: schoff@NISC.NYSER.NET

James R. Davin
MIT Laboratory for Computer Science, NE43-507
545 Technology Square
Cambridge, MA 02139

Phone: (617) 253-6020
EMail: jrd@ptt.lcs.mit.edu

[1] Cerf, V., "IAB Recommendations for the Development of Internet Network Management Standards", RFC 1052, IAB, April 1988.

[2] Rose, M., and K. McCloghrie, "Structure and Identification of Management Information for TCP/IP-based internets", RFC 1065, TWG, August 1988.

[3] McCloghrie, K., and M. Rose, "Management Information Base for Network Management of TCP/IP-based internets", RFC 1066, TWG, August 1988.

[4] Cerf, V., "Report of the Second Ad Hoc Network Management Review Group", RFC 1109, IAB, August 1989.

[5] Rose, M., and K. McCloghrie, "Structure and Identification of Management Information for TCP/IP-based Internets", RFC 1155, Performance Systems International and Hughes LAN Systems, May 1990.

[6] McCloghrie, K., and M. Rose, "Management Information Base for Network Management of TCP/IP-based Internets", RFC 1156, Hughes LAN Systems and Performance Systems International, May 1990.

[7] Case, J., M. Fedor, M. Schoffstall, and J. Davin, "A Simple Network Management Protocol", Internet Engineering Task Force working note, Network Information Center, SRI International, Menlo Park, California, March 1988.

[8] Davin, J., J. Case, M. Fedor, and M. Schoffstall, "A Simple Gateway Monitoring Protocol", RFC 1028, Proteon, University of Tennessee at Knoxville, Cornell University, and Rensselaer Polytechnic Institute, November 1987.

[9] Information processing systems - Open Systems Interconnection, "Specification of Abstract Syntax Notation One (ASN.1)", International Organization for Standardization, International Standard 8824, December 1987.

[10] Information processing systems - Open Systems Interconnection, "Specification of Basic Encoding Rules for Abstract Notation One (ASN.1)", International Organization for Standardization, International Standard 8825, December 1987.

[11] Postel, J., "User Datagram Protocol", RFC 768, USC/Information Sciences Institute, November 1980.

RFC-1160 The Internet Activities Board

V. Cerf
NRI
May 1990

Status of this Memo

This RFC provides a history and description of the Internet Activities Board (IAB) and its subsidiary organizations. This memo is for informational use and does not constitute a standard. This is a revision of RFC 1120. Distribution of this memo is unlimited.

RFC-1160 The Internet Activities Board

Introduction

In 1968, the U.S. Defense Advanced Research Projects Agency (DARPA) initiated an effort to develop a technology which is now known as packet switching. This technology had its roots in message switching methods, but was strongly influenced by the development of low-cost minicomputers and digital telecommunications techniques during the mid-1960's [BARAN 64, ROBERTS 70, HEART 70, ROBERTS 78]. A very useful survey of this technology can be found in [IEEE 78].

During the early 1970's, DARPA initiated a number of programs to explore the use of packet switching methods in alternative media including mobile radio, satellite and cable [IEEE 78]. Concurrently, Xerox Palo Alto Research Center (PARC) began an exploration of packet switching on coaxial cable which ultimately led to the development of Ethernet local area networks [METCALFE 76].

The successful implementation of packet radio and packet satellite technology raised the question of interconnecting ARPANET with other types of packet nets. A possible solution to this problem was proposed by Cerf and Kahn [CERF 74] in the form of an internetwork protocol and a set of gateways to connect the different networks. This solution was further developed as part of a research program in internetting sponsored by DARPA and resulted in a collection of computer communications protocols based on the original Transmission Control Protocol (TCP) and its lower level counterpart, Internet Protocol (IP). Together, these protocols, along with many others developed during the course of the research, are referred to as the TCP/IP Protocol Suite [RFC 1250, LEINER 85, POSTEL 85, CERF 82, CLARK 86].

In the early stages of the Internet research program, only a few researchers worked to develop and test versions of the internet protocols. Over time, the size of this activity increased until, in 1979, it was necessary to form an informal committee to guide the technical evolution of the protocol suite. This group was called the Internet Configuration Control Board (ICCB) and was established by Dr. Vinton Cerf who was then the DARPA program manager for the effort. Dr. David C. Clark of the Laboratory for Computer Science at Massachusetts Institute of Technology was named the chairman of this committee.

In January, 1983, the Defense Communications Agency, then responsible for the operation of the ARPANET, declared the TCP/IP protocol suite to be standard for the ARPANET and all systems on the network converted from the earlier Network Control Program (NCP) to TCP/IP. Late that year, the ICCB was reorganized by Dr. Barry Leiner, Cerf's successor at DARPA, around a series of task forces considering different technical aspects of internetting. The re-organized group was named the Internet Activities Board.

As the Internet expanded, it drew support from U.S. Government organizations including DARPA, the National Science Foundation (NSF), the Department of Energy (DOE) and the National Aeronautics and Space Administration (NASA). Key managers in these organizations, responsible for computer networking research and development, formed an informal Federal Research Internet Coordinating Committee (FRICC) to coordinate U.S. Government support for and development and use of the Internet system. The FRICC sponsored most of the U.S. research on internetting, including support for the Internet Activities Board and its subsidiary organizations.

In 1990, the FRICC was reorganized as part of a larger initiative sponsored by the networking subcommittee of the Federal Coordinating Committee on Science, Engineering and Technology (FCCSET). The reorganization created the Federal Networking Council (FNC) and its Working Groups. The membership of the FNC included all the former FRICC members and many other U.S. Government representatives. The first chairman of the FNC is Dr. Charles Brownstein of the National Science Foundation. The FNC is the Federal

Government's body for coordinating the agencies that support the Internet. It provides liaison to the Office of Science and Technology Policy (headed by the President's Science Advisor) which is responsible for setting science and technology policy affecting the Internet. It endorses and employs the existing planning and operational activities of the community-based bodies that have grown up to manage the Internet in the United States. The FNC plans to involve user and supplier communities through creation of an external advisory board and will coordinate Internet activities with other Federal initiatives ranging from the Human Genome and Global Change programs to educational applications. The FNC has also participated in planning for the creation of a National Research and Education Network in the United States.

At the international level, a Coordinating Committee for Intercontinental Research Networks (CCIRN) has been formed which includes the U.S. FNC and its counterparts in North America and Europe. Co-chaired by the executive directors of the FNC and the European Association of Research Networks (RARE), the CCIRN provides a forum for cooperative planning among the principal North American and European research networking bodies.

RFC-1160 The Internet Activities Board

Internet Activities Board

The Internet Activities Board (IAB) is the coordinating committee for Internet design, engineering and management. The Internet is a collection of over two thousand of packet switched networks located principally in the U.S., but also in many other parts of the world, all interlinked and operating using the protocols of the TCP/IP protocol suite. The IAB is an independent committee of researchers and professionals with a technical interest in the health and evolution of the Internet system. Membership changes with time to adjust to the current realities of the research interests of the participants, the needs of the Internet system and the concerns of constituent members of the Internet.

IAB members are deeply committed to making the Internet function effectively and evolve to meet a large scale, high speed future. New members are appointed by the chairman of the IAB, with the advice and consent of the remaining members. The chairman serves a term of two years and is elected by the members of the IAB. The IAB focuses on the TCP/IP protocol suite, and extensions to the Internet system to support multiple protocol suites.

The IAB has two principal subsidiary task forces:

1) Internet Engineering Task Force (IETF)
2) Internet Research Task Force (IRTF)

Each of these Task Forces is led by a chairman and guided by a Steering Group which reports to the IAB through its chairman. Each task force is organized, by the chairman, as required, to carry out its charter. For the most part, a collection of Working Groups carries out the work program of each Task Force.

All decisions of the IAB are made public. The principal vehicle by which IAB decisions are propagated to the parties interested in the Internet and its TCP/IP protocol suite is the Request for Comment (RFC) note series. The archival RFC series was initiated in 1969 by Dr. Stephen D. Crocker as a means of documenting the development of the original ARPANET protocol suite [RFC 1000]. The editor-in-chief of this series, Dr. Jonathan B. Postel, has maintained the quality of and managed the archiving of this series since its inception. A small proportion of the RFCs document Internet standards. Most of them are intended to stimulate comment and discussion. The small number which document standards are especially marked in a "status" section to indicate the special status of the document. An RFC summarizing the status of all standard RFCs is published regularly [RFC 1250].

RFCs describing experimental protocols, along with other submissions whose intent is merely to inform, are typically submitted directly to the RFC editor. A Standard Protocol starts out as a Proposed Standard and may be promoted to Draft Standard and finally Standard after suitable review, comment, implementation and testing.

Prior to publication of a Proposed Standard RFC, it is made available for comment through an on-line Internet-Draft directory. Typically, these Internet-Drafts are working documents of the IAB or of the working groups of the Internet Engineering and Research Task Forces. Internet-Drafts are either submitted to the RFC editor for publication or discarded within 3-6 months. Prior to promotion to Draft Standard or Standard, an Internet-Draft publication and review cycle may be initiated if significant changes to the RFC are contemplated.

The IAB performs the following functions:

1) Sets Internet Standards,
2) Manages the RFC publication process,
3) Reviews the operation of the IETF and IRTF,

4) Performs strategic planning for the Internet, identifying long-range problems and opportunities,

5) Acts as an international technical policy liaison and representative for the Internet community, and

6) Resolves technical issues which cannot be treated within the IETF or IRTF frameworks.

To supplement its work via electronic mail, the IAB meets quarterly to review the condition of the Internet, to review and approve proposed changes or additions to the TCP/IP suite of protocols, to set technical development priorities, to discuss policy matters which may need the attention of the Internet sponsors, and to agree on the addition or retirement of IAB members and on the addition or retirement of task forces reporting to the IAB. Typically, two of the quarterly meetings are by means of video teleconferencing (provided, when possible, through the experimental Internet packet video-conferencing system). The minutes of the IAB meetings are published in the Internet Monthly on-line report.

The IAB membership is currently as follows:

Vinton Cerf/CNRI	Chairman
Robert Braden/USC-ISI	Executive Director
David Clark/MIT-LCS	IRTF Chairman
Phillip Gross/CNRI	IETF Chairman
Jonathan Postel/USC-ISI	RFC Editor
Hans-Werner Braun/Merit	Member
Lyman Chapin/DG	Member
Stephen Kent/BBN	Member
Anthony Lauck/Digital	Member
Barry Leiner/RIACS	Member
Daniel Lynch/Interop, Inc.	Member

RFC-1160 The Internet Activities Board

The Internet Engineering Task Force

The Internet has grown to encompass a large number of widely geographically dispersed networks in academic and research communities. It now provides an infrastructure for a broad community with various interests. Moreover, the family of Internet protocols and system components has moved from experimental to commercial development. To help coordinate the operation, management and evolution of the Internet, the IAB established the Internet Engineering Task Force (IETF). The IETF is chaired by Mr. Phillip Gross and managed by its Internet Engineering Steering Group (IESG). The IAB has delegated to the IESG the general responsibility for making the Internet work and for the resolution of all short- and mid-range protocol and architectural issues required to make the Internet function effectively.

The charter of the IETF includes:

1) Responsibility for specifying the short and mid-term Internet protocols and architecture and recommending standards for IAB approval.
2) Provision of a forum for the exchange of information within the Internet community.
3) Identification of pressing and relevant short- to mid-range operational and technical problem areas and convening of Working Groups to explore solutions.

The Internet Engineering Task Force is a large open community of network designers, operators, vendors, and researchers concerned with the Internet and the Internet protocol suite. It is organized around a set of eight technical areas, each managed by a technical area director. In addition to the IETF Chairman, the area directors make up the IESG membership. Each area director has primary responsibility for one area of Internet engineering activity, and hence for a subset of the IETF Working Groups. The area directors have jobs of critical importance and difficulty and are selected not only for their technical expertise but also for their managerial skills and judgment. At present, the eight technical areas and chairs are:

1) Applications	Russ Hobby/UC-Davis
2) Host and User Services	Craig Partridge/BBN
3) Internet Services	Noel Chiappa/Consultant
4) Routing	Robert Hinden/BBN
5) Network Management	David Crocker/DEC
6) OSI Integration	Ross Callon/DEC and Robert Hagens/UWisc.
7) Operations	Phill Gross/CNRI (Acting)
8) Security	Steve Crocker/TIS

The work of the IETF is performed by subcommittees known as Working Groups. There are currently more than 40 of these. Working Groups tend to have a narrow focus and a lifetime bounded by completion of a specific task, although there are exceptions. The IETF is a major source of proposed protocol standards, for final approval by the IAB. The IETF meets quarterly and extensive minutes of the plenary proceedings as well as reports from each of the working groups are issued by the IAB Secretariat at the Corporation for National Research Initiatives.

The Internet Research Task Force

To promote research in networking and the development of new technology, the IAB established the Internet Research Task Force (IRTF).

In the area of network protocols, the distinction between research and engineering is not always clear, so there will sometimes be overlap between activities of the IETF and the IRTF. There is, in fact, considerable overlap in membership between the two groups. This overlap is regarded as vital for cross-fertilization and technology transfer. In general, the distinction between research and engineering is one of viewpoint and sometimes (but not always) time-frame. The IRTF is generally more concerned with understanding than with products or standard protocols, although specific experimental protocols may have to be developed, implemented and tested in order to gain understanding.

The IRTF is a community of network researchers, generally with an Internet focus. The work of the IRTF is governed by its Internet Research Steering Group (IRSG). The chairman of the IRTF and IRSG is David Clark. The IRTF is organized into a number of Research Groups (RGs) whose chairs of these are appointed by the chairman of the IRSG. The RG chairs and others selected by the IRSG chairman serve on the IRSG. These groups typically have 10 to 20 members, and each covers a broad area of research, pursuing specific topics, determined at least in part by the interests of the members and by recommendations of the IAB.

The current members of the IRSG are as follows:

David Clark/MIT LCS	Chairman
Robert Braden/USC-ISI	End-to-End Services
Douglas Comer/PURDUE	Member-at-Large
Deborah Estrin/USC	Autonomous Networks
Stephen Kent/BBN	Privacy and Security
Keith Lantz/Consultant	Collaboration Technology
David Mills/UDEL	Member-at-Large

RFC-1160 The Internet Activities Board

The Near-term Agenda of the IAB

There are seven principal foci of IAB attention for the period 1989 - 1990:

1) Operational Stability
2) User Services
3) OSI Coexistence
4) Testbed Facilities
5) Security
6) Getting Big
7) Getting Fast

Operational stability of the Internet is a critical concern for all of its users. Better tools are needed for gathering operational data, to assist in fault isolation at all levels and to analyze the performance of the system. Opportunities abound for increased cooperation among the operators of the various Internet components [RFC 1109]. Specific, known problems should be dealt with, such as implementation deficiencies in some versions of the BIND domain name service resolver software. To the extent that the existing Exterior Gateway Protocol (EGP) is only able to support limited topologies, constraints on topological linkages and allowed transit paths should be enforced until a more general Inter-Autonomous System routing protocol can be specified. Flexiblity for Internet implementation would be enhanced by the adoption of a common internal gateway routing protocol by all vendors of internet routers. A major effort is recommended to achieve conformance to the Host Requirements RFCs which were published in the fourth quarter of calendar 1989.

Among the most needed user services, the White Pages (electronic mailbox directory service) seems the most pressing. Efforts should be focused on widespread deployment of these capabilities in the Internet by mid-1990. The IAB recommends that existing white pages facilities and newer ones, such as X.500, be populated with up-to- date user information and made accessible to Internet users and users of other systems (e.g., commercial email carriers) linked to the Internet. Connectivity with commercial electronic mail carriers should be vigorously pursued, as well as links to other network research communities in Europe and the rest of the world.

Development and deployment of privacy-enhanced electronic mail software should be accelerated in 1990 after release of public domain software implementing the private electronic mail standards [RFC 1113, RFC 1114 and RFC 1115]. Finally, support for new or enhanced applications such as computer-based conferencing, multi-media messaging and collaboration support systems should be developed.

The National Network Testbed (NNT) resources planned by the FRICC should be applied to support conferencing and collaboration protocol development and application experiments and to support multi-vendor router interoperability testing (e.g., interior and exterior routing, network management, multi-protocol routing and forwarding).

With respect to growth in the Internet, architectural attention should be focused on scaling the system to hundreds of millions of users and hundreds of thousands of networks. The naming, addressing, routing and navigation problems occasioned by such growth should be analyzed. Similarly, research should be carried out on analyzing the limits to the existing Internet architecture, including the ability of the present protocol suite to cope with speeds in the gigabit range and latencies varying from microseconds to seconds in duration.

The Internet should be positioned to support the use of OSI protocols by the end of 1990 or

sooner, if possible. Provision for multi- protocol routing and forwarding among diverse vendor routes is one important goal. Introduction of X.400 electronic mail services and interoperation with RFC 822/SMTP [RFC 822, RFC 821, RFC 987, RFC 1026, and RFC 1148] should be targeted for 1990 as well. These efforts will need to work in conjunction with the White Pages services mentioned above. The IETF, in particular, should establish liaison with various OSI working groups (e.g., at NIST, RARE, Network Management Forum) to coordinate planning for OSI introduction into the Internet and to facilitate registration of information pertinent to the Internet with the various authorities responsible for OSI standards in the United States.

Finally, with respect to security, a concerted effort should be made to develop guidance and documentation for Internet host managers concerning configuration management, known security problems (and their solutions) and software and technologies available to provide enhanced security and privacy to the users of the Internet.

Author's Address

Vinton G. Cerf
Corporation for National Research Initiatives
1895 Preston White Drive, Suite 100
Reston, VA 22091

Phone: (703) 620-8990

EMail: VCERF@NRI.RESTON.VA.US

[BARAN 64] Baran, P., et al, "On Distributed Communications", Volumes I-XI, RAND Corporation Research Documents, August 1964.

[CERF 74] Cerf V., and R. Kahn, "A Protocol for Packet Network Interconnection", IEEE Trans. on Communications, Vol. COM-22, No. 5, pp. 637-648, May 1974.

[CERF 82] Cerf V., and E. Cain, "The DoD Internet Protocol Architecture", Proceedings of the SHAPE Technology Center Symposium on Interoperability of Automated Data Systems, November 1982. Also in Computer Networks and ISDN, Vol. 17, No. 5, October 1983.

[CLARK 86] Clark, D., "The Design Philosophy of the DARPA Internet protocols", Proceedings of the SIGCOMM '88 Symposium, Computer Communications Review, Vol. 18, No. 4, pp. 106-114, August 1988.

[HEART 70] Heart, F., Kahn, R., Ornstein, S., Crowther, W., and D. Walden, "The Interface Message Processor for the ARPA Computer Network", AFIPS Conf. Proc. 36, pp. 551-567, June 1970.

[IEEE 78] Kahn, R. (Guest Editor), Uncapher, K. and H. Van Trees (Associate Guest Editors), Proceedings of the IEEE, Special Issue on Packet Communication Networks, Volume 66, No. 11, pp. 1303-1576, November 1978.

[IEEE 87] Leiner, B. (Guest Editor), Nielson, D., and F. Tobagi (Associate Guest Editors), Proceedings of the IEEE, Special Issue on Packet Radio Networks, Volume 75, No. 1, pp. 1-272, January 1987.

[LEINER 85] Leiner, B., Cole, R., Postel, J., and D. Mills, "The DARPA Protocol Suite", IEEE INFOCOM 85, Washington, D.C., March 1985. Also in IEEE Communications Magazine, March 1985.

[METCALFE 76] Metcalfe, R., and D. Boggs, "Ethernet:

[POSTEL85] Postel, J. "Internetwork Applications using the DARPA Protocol Suite", IEEE INFOCOM 85, Washington, D.C., March 1985.

[ROBERTS 70] Roberts, L., and B. Wessler, "Computer Network Development to Achieve Resource Sharing", pp. 543-549, Proc. SJCC 1970.

[ROBERTS 78] Roberts, L., "Evolution of Packet Switching", Proc. IEEE, Vol. 66, No. 11, pp. 1307-1313, November 1978.

RFC-1171
The Point-to-Point Protocol for the Transmissionof Multi-Protocol Datagrams Over Point-to-Point Links

D. Perkins, CMU, July 1990

Obsoletes: RFC 1134

Status of this Memo

This RFC specifies an IAB standards track protocol for the Internet community. Please refer to the current edition of the "IAB Official Protocol Standards" for the standardization state and status of this protocol.

This proposal is the product of the Point-to-Point Protocol Working Group of the Internet Engineering Task Force (IETF). Comments on this memo should be submitted to the IETF Point-to-Point Protocol Working Group chair.

Security issues are not discussed in this memo. Distribution of this memo is unlimited.

Abstract

The Point-to-Point Protocol (PPP) provides a method for transmitting datagrams over serial point-to-point links. PPP is composed of three parts:

1. A method for encapsulating datagrams over serial links.
2. An extensible Link Control Protocol (LCP).
3. A family of Network Control Protocols (NCP) for establishing and configuring different network-layer protocols.

This document defines the encapsulation scheme, the basic LCP, and an NCP for establishing and configuring the Internet Protocol (IP) (called the IP Control Protocol, IPCP).

The options and facilities used by the LCP and the IPCP are defined in separate documents. Control protocols for configuring and utilizing other network-layer protocols besides IP (e.g., DECNET, OSI) are expected to be developed as needed.

Introduction

Motivation

In the last few years, the Internet has seen explosive growth in the number of hosts supporting TCP/IP. The vast majority of these hosts are connected to Local Area Networks (LANs) of various types, Ethernet being the most common. Most of the other hosts are connected through Wide Area Networks (WANs) such as X.25 style Public Data Networks (PDNs). Relatively few of these hosts are connected with simple point-to-point (i.e., serial) links. Yet, point-to-point links are among the oldest methods of data communications and almost every host supports point-to-point connections. For example, asynchronous RS-232-C interfaces are essentially ubiquitous.

One reason for the small number of point-to-point IP links is the lack of a standard encapsulation protocol. There are plenty of non-standard (and at least one defacto standard) encapsulation protocols available, but there is not one which has been agreed upon as an Internet Standard. By contrast, standard encapsulation schemes do exist for the transmission of datagrams over most popular LANs.

One purpose of this memo is to remedy this problem. But even more importantly, the Point-to-Point Protocol proposes more than just an encapsulation scheme. Point-to-Point links tend to exacerbate many problems with the current family of network protocols. For instance, assignment and management of IP addresses, which is a problem even in LAN environments, is especially difficult over circuit switched point-to-point circuits (e.g., dialups).

Some additional issues addressed by this specification of PPP include asynchronous (start/stop) and bit-oriented synchronous encapsulation, network protocol multiplexing, link configuration, link quality testing, error detection, and option negotiation for such capabilities as network-layer address negotiation and data compression negotiation.

PPP addresses these issues by providing an extensible Link Control Protocol (LCP) and a family of Network Control Protocols (NCP) to negotiate optional configuration parameters and facilities.

RFC-1171 The Point-to-Point Protocol: Introduction

Overview of PPP

PPP has three main components:

1. A method for encapsulating datagrams over serial links. PPP uses HDLC as a basis for encapsulating datagrams over point-to-point links. At this time, PPP specifies the use of asynchronous or synchronous duplex circuits, either dedicated or circuit switched.
2. An extensible Link Control Protocol (LCP) to establish, configure, and test the data-link connection.
3. A family of Network Control Protocols (NCP) for establishing and configuring different network-layer protocols. PPP is designed to allow the simultaneous use of multiple network-layer protocols.

In order to establish communications over a point-to-point link, the originating PPP would first send LCP packets to configure and test the data link. After the link has been establish and optional facilities have been negotiated as needed by the LCP, the originating PPP would send NCP packets to choose and configure one or more network-layer protocols. Once each of the chosen network-layer protocols has been configured, datagrams from each network-layer protocol can be sent over the link.

The link will remain configured for communications until explicit LCP or NCP packets close the link down, or until some external event occurs (e.g., inactivity timer expires or user intervention).

Organization of the document

This memo is divided into several sections. Section 2 discusses the physical-layer requirements of PPP. Section 3 describes the Data Link Layer including the PPP frame format and data link encapsulation scheme. Section 4 specifies the LCP including the connection establishment and option negotiation procedures. Section 5 specifies the IP Control Protocol (IPCP), which is the NCP for the Internet Protocol, and describes the encapsulation of IP datagrams within PPP packets. Appendix A summarizes important features of asynchronous HDLC, and Appendix B describes an efficient table-lookup algorithm for fast Frame Check Sequence (FCS) computation.

Physical Layer Requirements

The Point-to-Point Protocol is capable of operating across any DTE/DCE interface (e.g., EIA RS-232-C, EIA RS-422, EIA RS-423 and CCITT V.35). The only absolute requirement imposed by PPP is the provision of a duplex circuit, either dedicated or circuit switched, which can operate in either an asynchronous (start/stop) or synchronous bit-serial mode, transparent to PPP Data Link Layer frames. PPP does not impose any restrictions regarding transmission rate, other than those imposed by the particular DTE/DCE interface in use.

PPP does not require the use of modem control signals, such as Request To Send (RTS), Clear To Send (CTS), Data Carrier Detect (DCD), and Data Terminal Ready (DTR). However, using such signals when available can allow greater functionality and performance.

The Data Link Layer

The Point-to-Point Protocol uses the principles, terminology, and frame structure of the International Organization For Standardization's (ISO) High-level Data Link Control (HDLC) procedures (ISO 3309-1979), as modified by ISO 3309:1984/PDAD1 "Addendum 1: Start/stop transmission". ISO 3309-1979 specifies the HDLC frame structure for use in synchronous environments. ISO 3309:1984/PDAD1 specifies proposed modifications to ISO 3309-1979 to allow its use in asynchronous environments.

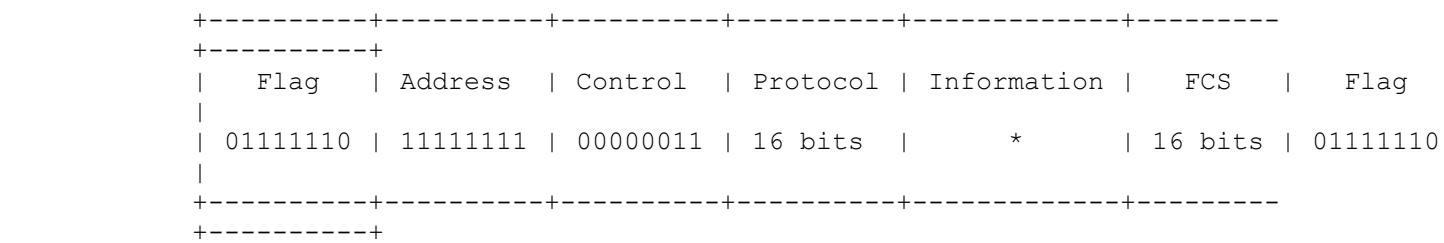
The PPP control procedures use the definitions and Control field encodings standardized in ISO 4335-1979 and ISO 4335-1979/Addendum 1-1979. The PPP frame structure is also consistent with CCITT Recommendation X.25 LAPB, since that too is based on HDLC.

Note: ISO 3309:1984/PDAD1 is a Proposed Draft standard. At present, it seems that ISO 3309:1984/PDAD1 is stable and likely to become an International Standard. Therefore, we feel comfortable about using it before it becomes an International Standard. The progress of this proposal should be tracked and encouraged by the Internet community.

The purpose of this memo is not to document what is already standardized in ISO 3309. We assume that the reader is already familiar with HDLC, or has access to a copy of ISO Standard 3309-1979 or CCITT X.25 Instead, this paper attempts to give a concise summary and point out specific options and features used by PPP. Since "Addendum 1: Start/stop transmission", is not yet standardized and widely available, it is summarized in Appendix A.

Frame Format

A summary of the standard PPP frame structure is shown below. The fields are transmitted from left to right.



Flag	Address	Control	Protocol	Information	FCS	Flag
01111110	11111111	00000011	16 bits	*	16 bits	01111110

This figure does not include start/stop bits (for asynchronous links) or any bits or octets inserted for transparency. When asynchronous links are used, all octets are transmitted with one start bit, eight bits of data, and one stop bit. There is no provision in either PPP or ISO 3309:1984/PDAD1 for seven bit asynchronous links.

To remain consistent with standard Internet practice, and avoid confusion for people used to reading RFCs, all binary numbers in the following descriptions are in Most Significant Bit to Least Significant Bit order, reading from left to right, unless otherwise indicated. Note that this is contrary to standard ISO and CCITT practice which orders bits as transmitted (i.e., network bit order). Keep this in mind when comparing this document with the international standards documents.

Flag Sequence

The Flag Sequence is a single octet and indicates the beginning or end of a frame. The Flag Sequence consists of the binary sequence 01111110 (hexadecimal 0x7e).

Address Field

The Address field is a single octet and contains the binary sequence 11111111 (hexadecimal 0xff), the All-Stations address. PPP does not assign individual station addresses. The All-Stations address should always be recognized and received. The use of other address lengths and values may be defined at a later time, or by prior agreement. Frames with unrecognized Addresses should be reported through the normal network management facility.

Control Field

The Control field is a single octet and contains the binary sequence 00000011 (hexadecimal 0x03), the Unnumbered Information (UI) command with the P/F bit set to zero. Frames with other Control field values should be silently discarded.

Protocol Field

The Protocol field is two octets and its value identifies the protocol encapsulated in the Information field of the frame. The most up-to-date values of the Protocol field are specified in the most recent "Assigned Numbers" RFC. Initial values are also listed below.

Protocol field values in the "cxxx" range identify datagrams as belonging to the Link Control Protocol (LCP) or associated protocols. Values in the "8xxx" range identify datagrams belonging to the family of Network Control Protocols (NCP). Values in the "0xxx" range identify the network protocol of specific datagrams.

This Protocol field is defined by PPP and is not a field defined by HDLC. However, the Protocol field is consistent with the ISO 3309 extension mechanism for Address fields. All Protocols MUST be odd; the least significant bit of the least significant octet MUST equal "1". Also, all Protocols MUST be assigned

such that the least significant bit of the most significant octet equals "0". Frames received which don't comply with these rules should be considered as having an unrecognized Protocol, and should be handled as specified by the LCP. The Protocol field is transmitted and received most significant octet first.

The Protocol field is initially assigned as follows:

Value (in hex)	Protocol
0001 to 001f	reserved (transparency inefficient)
0021	Internet Protocol
0023	* OSI Network Layer
0025	* Xerox NS IDP
0027	* DECnet Phase IV
0029	* Appletalk
002b	* Novell IPX
002d	* Van Jacobson Compressed TCP/IP 1
002f	* Van Jacobson Compressed TCP/IP 2
8021	Internet Protocol Control Protocol
8023	* OSI Network Layer Control Protocol
8025	* Xerox NS IDP Control Protocol
8027	* DECnet Phase IV Control Protocol
8029	* Appletalk Control Protocol
802b	* Novell IPX Control Protocol
802d	* Reserved
802f	* Reserved
c021	Link Control Protocol
c023	* User/Password Authentication Protocol

* Reserved for future use; not described in this document.

Information Field

The Information field is zero or more octets. The Information field contains the datagram for the protocol specified in the Protocol field. The end of the Information field is found by locating the closing Flag Sequence and allowing two octets for the Frame Check Sequence field. The default maximum length of the Information field is 1500 octets. By prior agreement, consenting PPP implementations may use other values for the maximum Information field length.

On transmission, the Information field may be padded with an arbitrary number of octets up to the maximum length. It is the responsibility of each protocol to disambiguate padding octets from real information.

Frame Check Sequence (FCS) Field

The Frame Check Sequence field is normally 16 bits (two octets). By prior agreement, consenting PPP implementations may use a 32-bit (four-octet) FCS for improved error detection.

The FCS field is calculated over all bits of the Address, Control, Protocol and Information fields not including any start and stop bits (asynchronous) and any bits (synchronous) or octets (asynchronous) inserted for transparency. This does not include the Flag Sequences or FCS field. The FCS is transmitted with the coefficient of the highest term first.

For more information on the specification of the FCS, see ISO 3309 or CCITT X.25.

Note: A fast, table-driven implementation of the 16-bit FCS algorithm is shown in Appendix B. This implementation is based on work by Perez, Morse, and LeVan.

Modifications to the Basic Frame Format

The Link Control Protocol can negotiate modifications to the standard PPP frame structure. However, modified frames will always be clearly distinguishable from standard frames.

RFC-1171 The Point-to-Point Protocol

The PPP Link Control Protocol (LCP)

The Link Control Protocol (LCP) provides a method of establishing, configuring, maintaining and terminating the point-to-point connection.

LCP goes through four distinct phases:

Phase 1: Link Establishment and Configuration Negotiation

Before any network-layer datagrams (e.g., IP) may be exchanged, LCP must first open the connection through an exchange of Configure packets. This exchange is complete, and the Open state entered, once a Configure-Ack packet (described below) has been both sent and received. Any non-LCP packets received before this exchange is complete are silently discarded.

It is important to note that LCP handles configuration only of the link; LCP does not handle configuration of individual network-layer protocols. In particular, all Configuration Parameters which are independent of particular network-layer protocols are configured by LCP. All Configuration Options are assumed to be at default values unless altered by the configuration exchange.

Phase 2: Link Quality Determination

LCP allows an optional Link Quality Determination phase following transition to the LCP Open state. In this phase, the link is tested to determine if the link quality is sufficient to bring up network-layer protocols. This phase is completely optional. LCP may delay transmission of network-layer protocol information until this phase is completed.

The procedure for Link Quality Determination is unspecified and may vary from implementation to implementation, or because of user-configured parameters, but only so long as the procedure doesn't violate other aspects of LCP. One suggested method is to use LCP Echo-Request and Echo-Reply packets.

What is important is that this phase may persist for any length of time. Therefore, implementations should avoid fixed timeouts when waiting for their peers to advance to phase 3.

Phase 3: Network-Layer Protocol Configuration Negotiation

Once LCP has finished the Link Quality Determination phase, network-layer protocols may be separately configured by the appropriate Network Control Protocols (NCP), and may be brought up and taken down at any time. If LCP closes the link, it informs the network-layer protocols so that they may take appropriate

Phase 4: Link Termination

LCP may terminate the link at any time. This will usually be done at the request of a human user, but may happen because of a physical event such as the loss of carrier, or the expiration of an idle-period timer.

The LCP Automaton

Overview

LCP is specified by a number of packet formats and a finite-state automaton. This section presents an overview of the LCP automaton, followed by a representation of it as both a state diagram and a state transition table.

There are three classes of LCP packets:

1. Link Establishment packets used to establish and configure a link, (e.g., Configure-Request, Configure-Ack, Configure-Nak and Configure-Reject)
2. Link Termination packets used to terminate a link, (e.g., Terminate-Request and Terminate-Ack)
3. Link Maintenance packets used to manage and debug a link, (e.g., Code-Reject, Protocol-Reject, Echo-Request, Echo-Reply and Discard-Request)

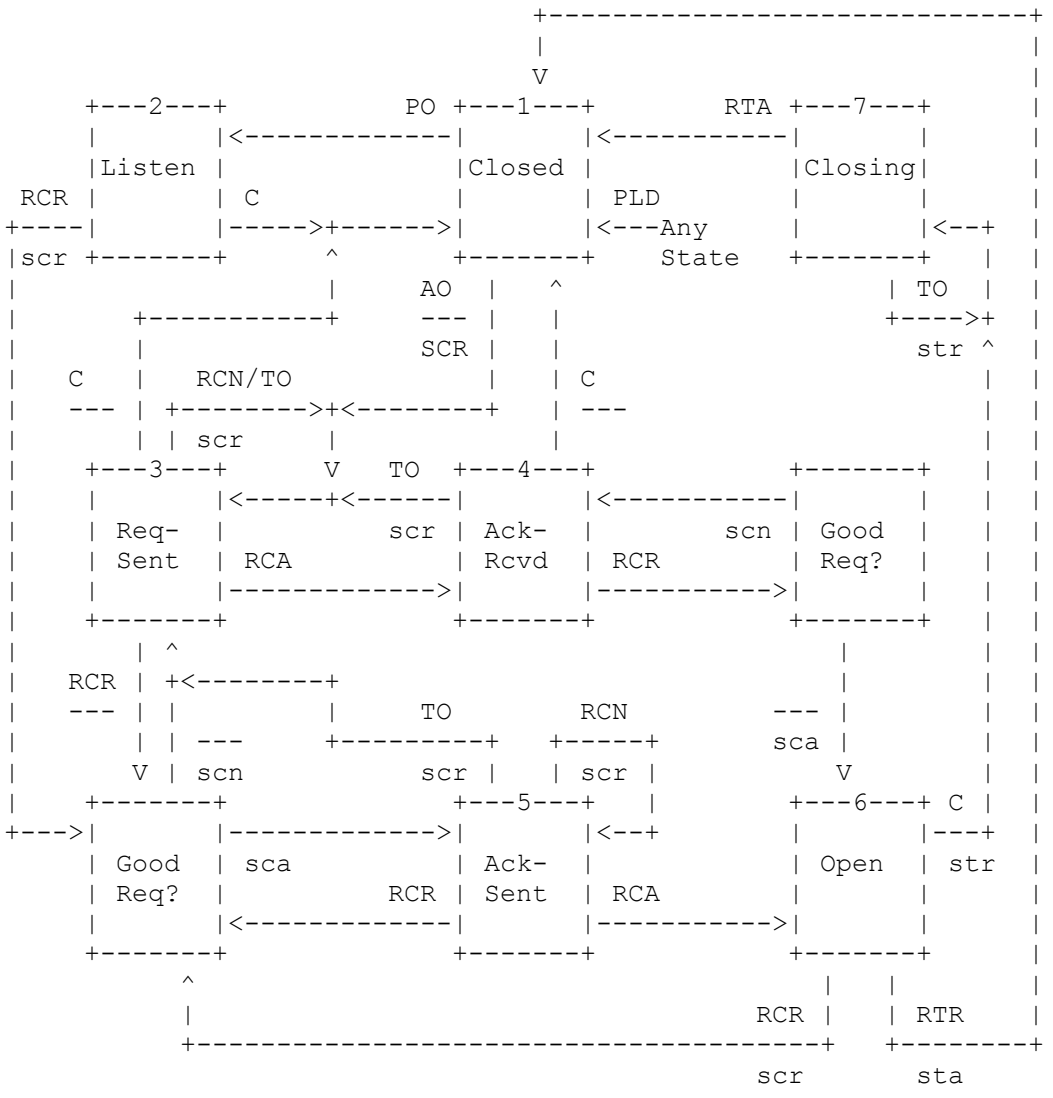
The finite-state automaton is defined by events, state transitions and actions. Events include receipt of external commands such as Open and Close, expiration of the Restart timer, and receipt of packets from a LCP peer. Actions include the starting of the Restart timer and transmission of packets.

State Diagram

The state diagram which follows describes the sequence of events for reaching agreement on Configuration Options (opening the PPP connection) and for later closing of the connection. The state machine is initially in the Closed state (1). Once the Open state (6) has been reached, both ends of the link have met the requirement of having both sent and received a Configure-Ack packet.

In the state diagram, events are shown above horizontal lines. Actions are shown below horizontal lines. Two types of LCP packets - Configure-Naks and Configure-Rejects - are not differentiated in the state diagram. As will be described later, these packets do indeed serve different, though similar, functions. However, at the level of detail of this state diagram, they always cause the same transition.

Since a more detailed specification of the LCP automaton is given in a state transition table in the following section, implementation should be done by consulting it rather than this state diagram.



Events

RCR - Receive-Configure-Request
RCA - Receive-Configure-Ack
RCN - Receive-Configure-Nak or Reject

Actions

scr - Send Configure-Request
sca - Send Configure-Ack
scn - Send Configure-Nak

RTR - Receive-Terminate-Req
RTA - Receive-Terminate-Ack
AO - Active-Open
PO - Passive-Open
C - Close
TO - Timeout
PLD - Physical-Layer-Down
or Reject
str - Send Terminate-Req
sta - Sent Terminate-Ack

State Transition Table

The complete state transition table follows. States are indicated horizontally, and events are read vertically. State transitions and actions are represented in the form action/new-state. Two actions caused by the same event are represented as action1&action2.

	State						
	1	2	3	4	5	6	7
Events	Closed	Listen	Req-Sent	Ack-Rcvd	Ack-Sent	Open	Closing
AO	scr/3	scr/3	3	4	5	6	scr/3
PO	2	2	2*	4	5	6	sta/3*
C	1	1	1*	1	str/7	str/7	7
TO	1	2	scr/3	scr/3	scr/3	6	str/7*
PLD	1	1	1	1	1	1	1
RCR+	sta/1	scr&sca/5	sca/5	sca/6	sca/5	scr&sca/5	7
RCR-	sta/1	scr&scn/3	scn/3	scn/4	scn/3	scr&scn/3	7
RCA	sta/1	sta/2	4	scr/3	6	scr/3	7
RCN	sta/1	sta/2	scr/3	scr/3	scr/5	scr/3	7
RTR	sta/1	sta/2	sta/3	sta/3	sta/3	sta/1	sta/7
RTA	1	2	3	3	3	1	1
RCJ	1	2	1	1	1	1	1
RUC	scj/1	scj/2	scj/1	scj/1	scj/1	scj/1	1 scj/7
RER	sta/1	sta/2	3	4	5	ser/6	7

Notes:

RCR+ - Receive-Configure-Request (Good)
RCR- - Receive-Configure-Request (Bad)
RCJ - Receive-Code-Reject
RUC - Receive-Unknown-Code
RER - Receive-Echo-Request
scj - Send-Code-Reject
ser - Send-Echo-Reply

* - Special attention necessary, see detailed text

Events

Transitions and actions in the LCP state machine are caused by events. Some events are caused by commands executed at the local end (e.g., Active-Open, Passive-Open, and Close), others are caused by the receipt of packets from the remote end (e.g., Receive-Configure-Request, Receive-Configure-Ack, Receive-Configure-Nak, Receive-Terminate-Request and Receive-Terminate-Ack), and still others are caused by the expiration of the Restart timer started as the result of other events (e.g., Timeout).

Following is a list of LCP events.

Active-Open (AO)

The Active-Open event indicates the local execution of an Active-Open command by the network administrator (human or program). When this event occurs, LCP should immediately attempt to open the connection by exchanging configuration packets with the LCP peer.

Passive-Open (PO)

The Passive-Open event is similar to the Active-Open event. However, instead of immediately exchanging configuration packets, LCP should wait for the peer to send the first packet. This will only happen after an Active-Open event in the LCP peer.

Close (C)

The Close event indicates the local execution of a Close command. When this event occurs, LCP should immediately attempt to close the connection.

Timeout (TO)

The Timeout event indicates the expiration of the LCP Restart timer. The LCP Restart timer is started as the result of other LCP events.

The Restart timer is used to time out transmissions of Configure-Request and Terminate-Request packets. Expiration of the Restart timer causes a Timeout event, which triggers the corresponding Configure-Request or Terminate-Request packet to be retransmitted. The Restart timer MUST be configurable, but should default to three (3) seconds.

Receive-Configure-Request (RCR)

The Receive-Configure-Request event occurs when a Configure-Request packet is received from the LCP peer. The Configure-Request packet indicates the desire to open a LCP connection and may specify Configuration Options. The Configure-Request packet is more fully described in a later section.

Receive-Configure-Ack (RCA)

The Receive-Configure-Ack event occurs when a valid Configure-Ack packet is received from the LCP peer. The Configure-Ack packet is a positive response to a Configure-Request packet.

Receive-Configure-Nak (RCN)

The Receive-Configure-Nak event occurs when a valid Configure-Nak or Configure-Reject packet is received from the LCP peer. The Configure-Nak and Configure-Reject packets are negative responses to a Configure-Request packet.

Receive-Terminate-Request (RTR)

The Receive-Terminate-Request event occurs when a Terminate-Request packet is received from the LCP peer. The Terminate-Request packet indicates the desire to close the LCP connection.

Receive-Terminate-Ack (RTA)

The Receive-Terminate-Ack event occurs when a Terminate-Ack packet is received from the LCP peer. The Terminate-Ack packet is a response to a Terminate-Request packet.

Receive-Code-Reject (RCJ)

The Receive-Code-Reject event occurs when a Code-Reject packet is received from the LCP peer. The Code-Reject packet communicates an error that immediately closes the connection.

Receive-Unknown-Code (RUC)

The Receive-Unknown-Code event occurs when an un-interpretable packet is received from the LCP peer. The Code-Reject packet is a response to an unknown packet.

Receive-Echo-Request (RER)

The Receive-Echo-Request event occurs when a Echo-Request, Echo-Reply, or Discard-Request packet is received from the LCP peer. The Echo-Reply packet is a response to a Echo-Request packet. There is no reply to a Discard-Request.

Physical-Layer-Down (PLD)

The Physical-Layer-Down event occurs when the Physical Layer indicates that it is down.

Actions

Actions in the LCP state machine are caused by events and typically indicate the transmission of packets and/or the starting or stopping of the Restart timer. Following is a list of LCP actions.

Send-Configure-Request (scr)

The Send-Configure-Request action transmits a Configure-Request packet. This indicates the desire to open a LCP connection with a specified set of Configuration Options. The Restart timer is started after the Configure-Request packet is transmitted, to guard against packet loss.

Send-Configure-Ack (sca)

The Send-Configure-Ack action transmits a Configure-Ack packet. This acknowledges the receipt of a Configure-Request packet with an acceptable set of Configuration Options.

Send-Configure-Nak (scn)

The Send-Configure-Nak action transmits a Configure-Nak or Configure-Reject packet, as appropriate. This negative response reports the receipt of a Configure-Request packet with an unacceptable set of Configuration Options. Configure-Nak packets are used to refuse a Configuration Option value, and to suggest a new, acceptable value. Configure-Reject packets are used to refuse all negotiation about a Configuration Option, typically because it is not recognized or implemented. The use of Configure-Nak vs. Configure-Reject is more fully described in the section on LCP Packet Formats.

Send-Terminate-Req (str)

The Send-Terminate-Request action transmits a Terminate-Request packet. This indicates the desire to close a LCP connection. The Restart timer is started after the Terminate-Request packet is transmitted, to guard against packet loss.

Send-Terminate-Ack (sta)

The Send-Terminate-Request action transmits a Terminate-Ack packet. This acknowledges the receipt of a Terminate-Request packet or otherwise confirms the belief that a LCP connection is Closed.

Send-Code-Reject (scj)

The Send-Code-Reject action transmits a Code-Reject packet. This indicates the receipt of an unknown type of packet. This is an unrecoverable error which causes immediate transitions to the Closed state on both ends of the link.

Send-Echo-Reply (ser)

The Send-Echo-Reply action transmits an Echo-Reply packet. This acknowledges the receipt of an Echo-Request packet.

States

Following is a more detailed description of each LCP state.

Closed (1)

The initial and final state is the Closed state. In the Closed state the connection is down and there is no attempt to open it; all connection requests from peers are rejected. Physical-Layer-Down events always cause an immediate transition to the Closed state.

There are two events which cause a transition out of the Closed state, Active-Open and Passive-Open. Upon an Active-Open event, a Configure-Request is transmitted, the Restart timer is started, and the Request-Sent state is entered. Upon a Passive-Open event, the Listen state is entered immediately. Upon receipt of any packet, with the exception of a Terminate-Ack, a Terminate-Ack is sent. Terminate-Acks are silently discarded to avoid creating a loop.

The Restart timer is not running in the Closed state.

The Physical Layer connection may be disconnected at any time when in the LCP Closed state.

Listen (2)

The Listen state is similar to the Closed state in that the connection is down and there is no attempt to open it. However, peer connection requests are no longer rejected.

Upon receipt of a Configure-Request, a Configure-Request is immediately transmitted and the Restart timer is started. The received Configuration Options are examined and the proper response is sent. If a Configure-Ack is sent, the Ack-Sent state is entered. Otherwise, if a Configure-Nak or Configure-Reject is sent, the Request-Sent state is entered. In either case, LCP exits its passive state, and begins to actively open the connection. Terminate-Ack packets are sent in response to either Configure-Ack or Configure-Nak packets,

The Restart timer is not running in the Listen state.

Request-Sent (3)

In the Request-Sent state an active attempt is made to open the connection. A Configure-Request has been sent and the Restart timer is running, but a Configure-Ack has not yet been received nor has one been sent.

Upon receipt of a Configure-Ack, the Ack-Received state is immediately entered. Upon receipt of a Configure-Nak or Configure-Reject, the Configure-Request Configuration Options are adjusted appropriately, a new Configure-Request is transmitted, and the Restart timer is restarted. Similarly, upon the expiration of the Restart timer, a new Configure-Request is transmitted and the Restart timer is restarted. Upon receipt of a Configure-Request, the Configuration Options are examined and if acceptable, a Configure-Ack is sent and the Ack-Sent state is entered. If the Configuration Options are unacceptable, a Configure-Nak or Configure-Reject is sent as appropriate.

Since there is an outstanding Configure-Request in the Request-Sent state, special care must be taken to implement the Passive-Open and Close events; otherwise, it is possible for the LCP peer to think the connection is open. Processing of either event should be postponed until there is reasonable assurance that the peer is not open. In particular, the Restart timer should be allowed to expire.

Ack-Received (4)

In the Ack-Received state, a Configure-Request has been sent and a Configure-Ack has been received. The Restart timer is still running since a Configure-Ack has not yet been transmitted.

Upon receipt of a Configure-Request with acceptable Configuration Options, a Configure-Ack is transmitted, the Restart timer is stopped and the Open state is entered. If the Configuration Options are unacceptable, a Configure-Nak or Configure-Reject is sent as appropriate. Upon the expiration of the

Restart timer, a new Configure-Request is transmitted, the Restart timer is restarted, and the state machine returns to the Request-Sent state.

Ack-Sent (5)

In the Ack-Sent state, a Configure-Ack and a Configure-Request have been sent but a Configure-Ack has not yet been received. The Restart timer is always running in the Ack-Sent state.

Upon receipt of a Configure-Ack, the Restart timer is stopped and the Open state is entered. Upon receipt of a Configure-Nak or Configure-Reject, the Configure-Request Configuration Options are adjusted appropriately, a new Configure-Request is transmitted, and the Restart timer is restarted. Upon the expiration of the Restart timer, a new Configure-Request is transmitted, the Restart timer is restarted, and the state machine returns to the Request-Sent state.

Open (6)

In the Open state, a connection exists and data may be communicated over the link. The Restart timer is not running in the Open state.

In normal operation, only two events cause transitions out of the Open state. Upon receipt of a Close command, a Terminate-Request is transmitted, the Restart timer is started, and the Closing state is entered. Upon receipt of a Terminate-Request, a Terminate-Ack is transmitted and the Closed state is entered. Upon receipt of an Echo-Request, an Echo-Reply is transmitted. Similarly, Echo-Reply and Discard-Request packets are silently discarded or processed as expected. All other events cause immediate transitions out of the Open state and should be handled as if the state machine were in the Listen state.

Closing (7)

In the Closing state, an active attempt is made to close the connection. A Terminate-Request has been sent and the Restart timer is running, but a Terminate-Ack has not yet been received.

Upon receipt of a Terminate-Ack, the Closed state is immediately entered. Upon the expiration of the Restart timer, a new Terminate-Request is transmitted and the Restart timer is restarted. After the Restart timer has expired Max-Restart times, this action may be skipped, and the Closed state may be entered. Max-Restart MUST be a configurable parameter.

Since there is an outstanding Terminate-Request in the Closing state, special care must be taken to implement the Passive-Open event; otherwise, it is possible for the LCP peer to think the connection is open. Processing of the Passive-Open event should be postponed until there is reasonable assurance that the peer is not open. In particular, the implementation should wait until the state machine would normally transition to the Closed state because of a Receive-Terminate-Ack event or Max-Restart Timeout events.

Loop Avoidance

Note that the protocol makes a reasonable attempt at avoiding Configuration Option negotiation loops. However, the protocol does NOT guarantee that loops will not happen. As with any negotiation, it is possible to configure two PPP implementations with conflicting policies that will never converge. It is also possible to configure policies which do converge, but which take significant time to do so. Implementors should keep this in mind and should implement loop detection mechanisms or higher level timeouts. If a timeout is implemented, it MUST be configurable.

Timers and Counters

There is one special timer used by LCP, the Restart timer. The Restart timer is used to time out transmissions of Configure-Request and Terminate-Request packets. Expiration of the Restart timer causes a Timeout event, and the corresponding Configure-Request or Terminate-Request packet retransmission. The Restart timer MUST be configurable, but should default to three (3) seconds.

There is one additional restart parameter, Max-Restarts. Max-Restarts indicates the number of packet retransmissions that are required before there is reasonable assurance that the link closed. Max-Restarts MUST also be configurable, but should default to ten (10) retransmissions.

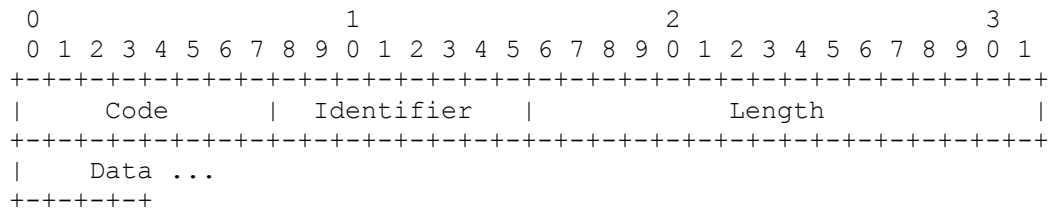
Packet Format

RFC-1171 The Point-to-Point Protocol: LCP Packet Format

Overview

Exactly one Link Control Protocol packet is encapsulated in the Information field of PPP Data Link Layer frames where the Protocol field indicates type hex c021 (Link Control Protocol).

A summary of the Link Control Protocol packet format is shown below. The fields are transmitted from left to right.



Code

The Code field is one octet and identifies the kind of LCP packet. LCP Codes are assigned as follows:

1	Configure-Request
2	Configure-Ack
3	Configure-Nak
4	Configure-Reject
5	Terminate-Request
6	Terminate-Ack
7	Code-Reject
8	Protocol-Reject
9	Echo-Request
10	Echo-Reply
11	Discard-Request

Identifier

The Identifier field is one octet and aids in matching requests and replies.

Length

The Length field is two octets and indicates the length of the LCP packet including the Code, Identifier, Length and Data fields. Octets outside the range of the Length field should be treated as Data Link Layer padding and should be ignored on reception.

Data

The Data field is zero or more octets as indicated by the Length field. The format of the Data field is determined by the Code field.

Regardless of which Configuration Options are enabled, all LCP packets are always sent in the full, standard form, as if no Configuration Options were enabled. This ensures that LCP Configure-Request packets are always recognizable even when one end of the link mistakenly believes the link to be Open.

This document describes Version 1 of the Link Control Protocol. In the interest of simplicity, there is no version field in the LCP packet. If a new version of LCP is necessary in the future, the intention is that a new Data Link Layer Protocol field value should be used to differentiate Version 1 LCP from all other versions. A correctly functioning Version 1 LCP implementation will always respond to unknown Protocols (including other versions) with an easily recognizable Version 1 packet, thus providing a deterministic fallback mechanism for implementations of other versions.

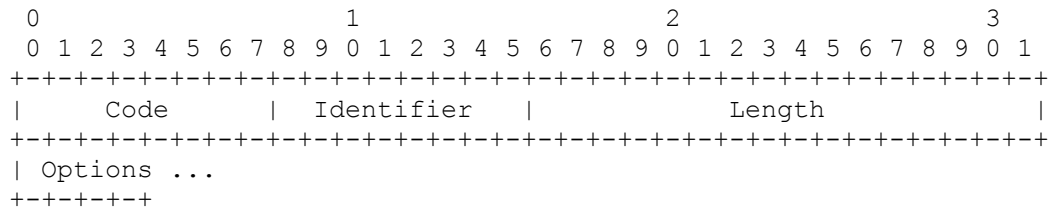
Configure-Request

Description

A LCP implementation wishing to open a connection MUST transmit a LCP packet with the Code field set to 1 (Configure-Request) and the Options field filled with any desired changes to the default link Configuration Options.

Upon reception of a Configure-Request, an appropriate reply MUST be transmitted.

A summary of the Configure-Request packet format is shown below. The fields are transmitted from left to right.



Code

1 for Configure-Request.

Identifier

The Identifier field should be changed on each transmission. On reception, the Identifier field should be copied into the Identifier field of the appropriate reply packet.

Options

The options field is variable in length and contains the list of zero or more Configuration Options that the sender desires to negotiate. All Configuration Options are always negotiated simultaneously. The format of Configuration Options is further described in a later section.

Configure-Ack

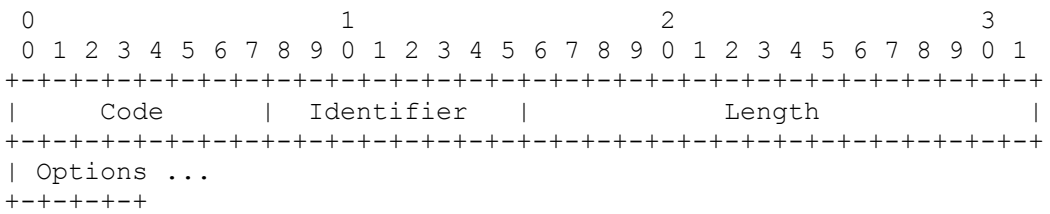
Description

If every Configuration Option received in a Configure-Request is both recognizable and acceptable, then a LCP implementation should transmit a LCP packet with the Code field set to 2 (Configure-Ack), the Identifier field copied from the received Configure-Request, and the Options field copied from the received Configure-Request. The acknowledged Configuration Options MUST NOT be reordered or modified in any way.

On reception of a Configure-Ack, the Identifier field must match that of the last transmitted Configure-Request, or the packet is invalid. Additionally, the Configuration Options in a Configure-Ack must match those of the last transmitted Configure-Request, or the packet is invalid. Invalid packets should be silently discarded.

Reception of a valid Configure-Ack indicates that all Configuration Options sent in the last Configure-Request are acceptable.

A summary of the Configure-Ack packet format is shown below. The fields are transmitted from left to right.



Code

2 for Configure-Ack.

Identifier

The Identifier field is a copy of the Identifier field of the Configure-Request which caused this Configure-Ack.

Options

The Options field is variable in length and contains the list of zero or more Configuration Options that the sender is acknowledging. All Configuration Options are always acknowledged simultaneously.

Configure-Nak

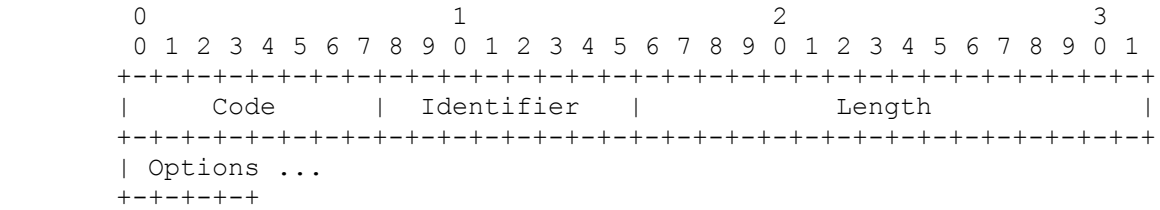
Description

If every element of the received Configuration Options is recognizable but some are not acceptable, then a LCP implementation should transmit a LCP packet with the Code field set to 3 (Configure-Nak), the Identifier field copied from the received Configure-Request and the Options field filled with only the unacceptable Configuration Options from the Configure-Request. All acceptable Configuration Options should be filtered out of the Configure-Nak, but otherwise the Configuration Options from the Configure-Request MUST NOT be reordered. Each of the nak'd Configuration Options MUST be modified to a value acceptable to the Configure-Nak sender. Finally, an implementation may be configured to require the negotiation of a specific option. If that option is not listed, then that option may be appended to the list of nak'd Configuration Options in order to request the remote end to list that option in its next Configure-Request packet. The appended option must include a value acceptable to the Configure-Nak sender.

On reception of a Configure-Nak, the Identifier field must match that of the last transmitted Configure-Request, or the packet is invalid and should be silently discarded.

Reception of a valid Configure-Nak indicates that a new Configure-Request should be sent with the Configuration Options modified as specified in the Configure-Nak.

A summary of the Configure-Nak packet format is shown below. The fields are transmitted from left to right.



Code

3 for Configure-Nak.

Identifier

The Identifier field is a copy of the Identifier field of the Configure-Request which caused this Configure-Nak.

Options

The Options field is variable in length and contains the list of zero or more Configuration Options that the sender is nak'ing. All Configuration Options are always nak'd simultaneously.

Configure-Reject

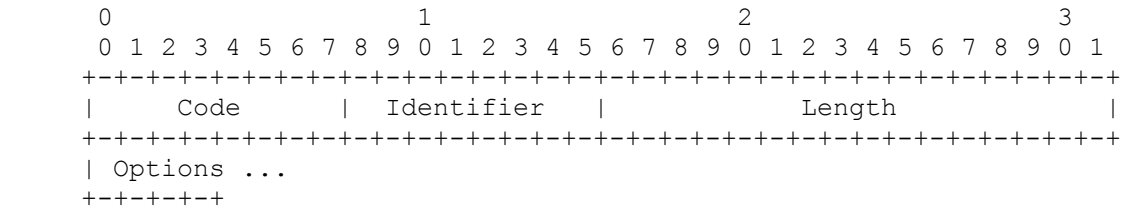
Description

If some Configuration Options received in a Configure-Request are not recognizable or are not acceptable for negotiation (as configured by a network manager), then a LCP implementation should transmit a LCP packet with the Code field set to 4 (Configure-Reject), the Identifier field copied from the received Configure-Request, and the Options field filled with only the unrecognized Configuration Options from the Configure-Request. All recognizable and negotiable Configuration Options must be filtered out of the Configure-Reject, but otherwise the Configuration Options MUST not be reordered.

On reception of a Configure-Reject, the Identifier field must match that of the last transmitted Configure-Request, or the packet is invalid. Additionally, the Configuration Options in a Configure-Reject must be a proper subset of those in the last transmitted Configure-Request, or the packet is invalid. Invalid packets should be silently discarded.

Reception of a Configure-Reject indicates that a new Configure-Request should be sent which does not include any of the Configuration Options listed in the Configure-Reject.

A summary of the Configure-Reject packet format is shown below. The fields are transmitted from left to right.



Code

4 for Configure-Reject.

Identifier

The Identifier field is a copy of the Identifier field of the Configure-Request which caused this Configure-Reject.

Options

The Options field is variable in length and contains the list of zero or more Configuration Options that the sender is rejecting. All Configuration Options are always rejected simultaneously.

RFC-1171 The Point-to-Point Protocol: LCP Packet Format

Terminate-Request and Terminate-Ack

Description

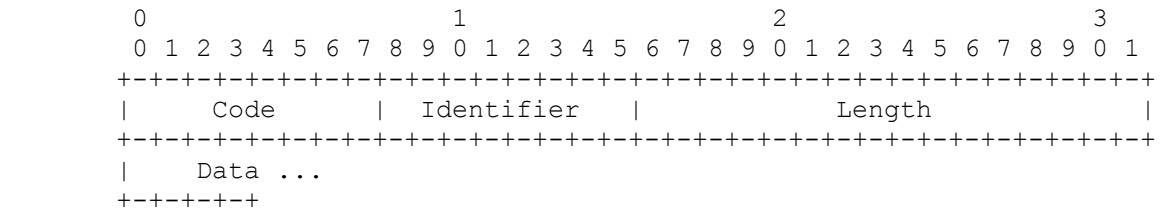
LCP includes Terminate-Request and Terminate-Ack Codes in order to provide a mechanism for closing a connection.

A LCP implementation wishing to close a connection should transmit a LCP packet with the Code field set to 5 (Terminate-Request) and the Data field filled with any desired data. Terminate-Request packets should continue to be sent until Terminate-Ack is received, the Physical Layer indicates that it has gone down, or a sufficiently large number have been transmitted such that the remote end is down with reasonable certainty.

Upon reception of a Terminate-Request, a LCP packet MUST be transmitted with the Code field set to 6 (Terminate-Ack), the Identifier field copied from the Terminate-Request packet, and the Data field filled with any desired data.

Reception of an unelicited Terminate-Ack indicates that the connection has been closed.

A summary of the Terminate-Request and Terminate-Ack packet formats is shown below. The fields are transmitted from left to right.



Code

5 for Terminate-Request;
6 for Terminate-Ack.

Identifier

The Identifier field is one octet and aids in matching requests and replies.

Data

The Data field is zero or more octets and contains uninterpreted data for use by the sender. The data may consist of any binary value and may be of any length from zero to the established maximum Information field length minus four.

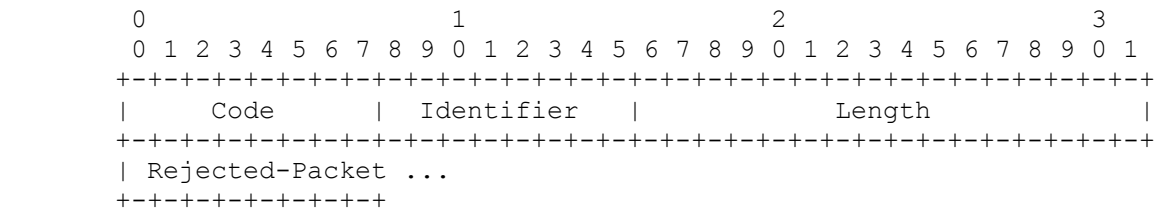
Code-Reject

Description

Reception of a LCP packet with an unknown Code indicates that one of the communicating LCP implementations is faulty or incomplete. This error MUST be reported back to the sender of the unknown Code by transmitting a LCP packet with the Code field set to 7 (Code-Reject), and the inducing packet copied to the Rejected-Packet field.

Upon reception of a Code-Reject, a LCP implementation should make an immediate transition to the Closed state, and should report the error, since it is unlikely that the situation can be rectified automatically.

A summary of the Code-Reject packet format is shown below. The fields are transmitted from left to right.



Code

7 for Code-Reject.

Identifier

The Identifier field is one octet and is for use by the transmitter.

Rejected-Packet

The Rejected-Packet field contains a copy of the LCP packet which is being rejected. It begins with the rejected Code field; it does not include any PPP Data Link Layer headers. The Rejected-Packet should be truncated to comply with the established maximum Information field length.

Protocol-Reject

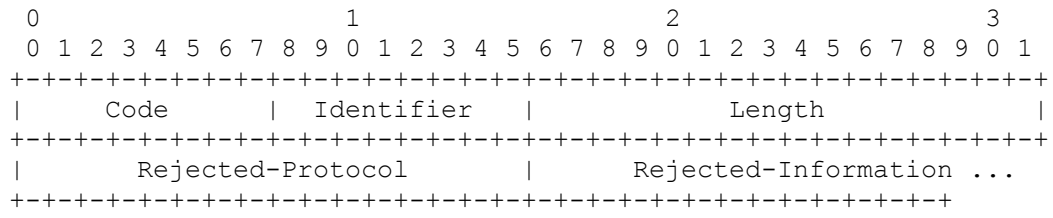
Description

Reception of a PPP frame with an unknown Data Link Layer Protocol indicates that the remote end is attempting to use a protocol which is unsupported at the local end. This typically occurs when the remote end attempts to configure a new, but unsupported protocol. If the LCP state machine is in the Open state, then this error MUST be reported back to the sender of the unknown protocol by transmitting a LCP packet with the Code field set to 8 (Protocol-Reject), the Rejected-Protocol field set to the received Protocol, and the Data field filled with any desired data.

Upon reception of a Protocol-Reject, a LCP implementation should stop transmitting frames of the indicated protocol.

Protocol-Reject packets may only be sent in the LCP Open state. Protocol-Reject packets received in any state other than the LCP Open state should be discarded and no further action should be taken.

A summary of the Protocol-Reject packet format is shown below. The fields are transmitted from left to right.



Code

8 for Protocol-Reject.

Identifier

The Identifier field is one octet and is for use by the transmitter.

Rejected-Protocol

The Rejected-Protocol field is two octets and contains the Protocol of the Data Link Layer frame which is being rejected.

Rejected-Information

The Rejected-Information field contains a copy from the frame which is being rejected. It begins with the Information field, and does not include any PPP Data Link Layer headers or the FCS. The Rejected-Information field should be truncated to comply with the established maximum Information field length.

RFC-1171 The Point-to-Point Protocol: LCP Packet Format

Echo-Request and Echo-Reply

Description

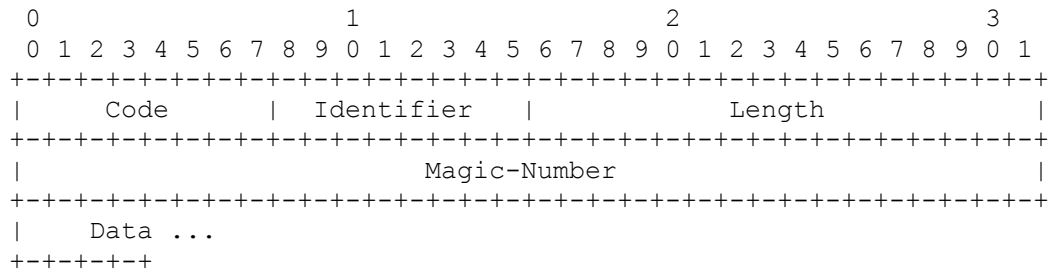
LCP includes Echo-Request and Echo-Reply Codes in order to provide a Data Link Layer loopback mechanism for use in exercising both directions of the link. This is useful as an aid in debugging, link quality determination, performance testing, and for numerous other functions.

An Echo-Request sender transmits a LCP packet with the Code field set to 9 (Echo-Request) and the Data field filled with any desired data, up to but not exceeding the receiver's established maximum Information field length minus eight.

Upon reception of an Echo-Request, a LCP packet MUST be transmitted with the Code field set to 10 (Echo-Reply), the Identifier field copied from the received Echo-Request, and the Data field copied from the Echo-Request, truncating as necessary to avoid exceeding the peer's established maximum Information field length.

Echo-Request and Echo-Reply packets may only be sent in the LCP Open state. Echo-Request and Echo-Reply packets received in any state other than the LCP Open state should be discarded and no further action should be taken.

A summary of the Echo-Request and Echo-Reply packet formats is shown below. The fields are transmitted from left to right.



Code

9 for Echo-Request;
10 for Echo-Reply.

Identifier

The Identifier field is one octet and aids in matching Echo-Requests and Echo-Replies.

Magic-Number

The Magic-Number field is four octets and aids in detecting loopbacked links. Unless modified by a Configuration Option, the Magic-Number MUST always be transmitted as zero and MUST always be ignored on reception. Further use of the Magic-Number is beyond the scope of this discussion.

Data

The Data field is zero or more octets and contains uninterpreted data for use by the sender. The data may consist of any binary value and may be of any length from zero to the established maximum Information field length minus eight.

RFC-1171 The Point-to-Point Protocol: LCP Packet Format

Discard-Request

Description

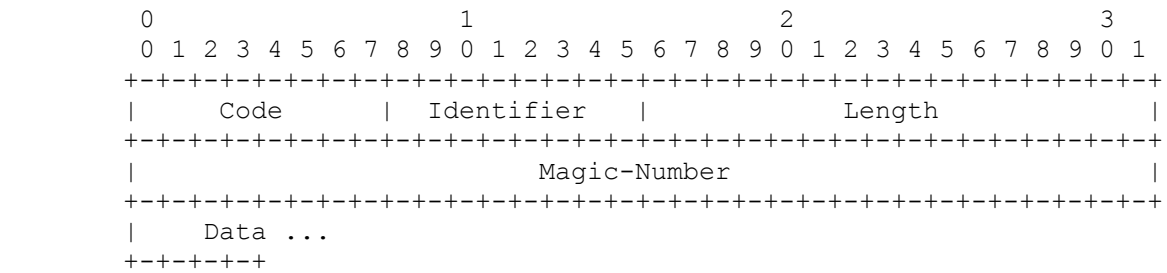
LCP includes a Discard-Request Code in order to provide a Data Link Layer data sink mechanism for use in exercising the local to remote direction of the link. This is useful as an aid in debugging, performance testing, and and for numerous other functions.

A discard sender transmits a LCP packet with the Code field set to 11 (Discard-Request) and the Data field filled with any desired data, up to but not exceeding the receiver's established maximum Information field length minus eight.

A discard receiver MUST simply throw away an Discard-Request that it receives.

Discard-Request packets may only be sent in the LCP Open state.

A summary of the Discard-Request packet formats is shown below. The fields are transmitted from left to right.



Code

11 for Discard-Request.

Identifier

The Identifier field is one octet and is for use by the Discard-Request transmitter.

Magic-Number

The Magic-Number field is four octets and aids in detecting loopbacked links. Unless modified by a configuration option, the Magic-Number MUST always be transmitted as zero and MUST always be ignored on reception. Further use of the Magic-Number is beyond the scope of this discussion.

Data

The Data field is zero or more octets and contains uninterpreted data for use by the sender. The data may consist of any binary value and may be of any length from zero to the established maximum Information field length minus four.

Configuration Options

LCP Configuration Options allow modifications to the standard characteristics of a point-to-point link to be negotiated. Negotiable modifications include such things as the maximum receive unit, async control character mapping, the link authentication method, etc. The Configuration Options themselves are described in separate documents. If a Configuration Option is not included in a Configure-Request packet, the default value for that Configuration Option is assumed.

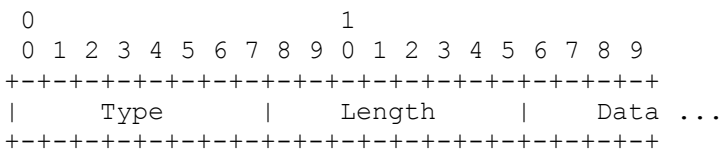
The end of the list of Configuration Options is indicated by the end of the LCP packet.

Unless otherwise specified, a specific Configuration Option should be listed no more than once in a Configuration Options list. Specific Configuration Options may override this general rule and may be listed more than once. The effect of this is Configuration Option specific and is specified by each such Configuration Option.

Also unless otherwise specified, all Configuration Options apply in a half-duplex fashion. When negotiated, they apply to only one direction of the link, typically in the receive direction when interpreted from the point of view of the Configure-Request sender.

Format

A summary of the Configuration Option format is shown below. The fields are transmitted from left to right.



Type

The Type field is one octet and indicates the type of Configuration Option. The most up-to-date values of the Type field are specified in the most recent "Assigned Numbers" RFC.

Length

The Length field is one octet and indicates the length of this Configuration Option including the Type, Length and Data fields. If a negotiable Configuration Option is received in a Configure-Request but with an invalid Length, a Configure-Nak should be transmitted which includes the desired Configuration Option with an appropriate Length and Data.

Data

The Data field is zero or more octets and indicates the value or other information for this Configuration Option. The format and length of the Data field is determined by the Type and Length fields.

A PPP Network Control Protocol (NCP) for IP

The IP Control Protocol (IPCP) is responsible for configuring, enabling, and disabling the IP protocol modules on both ends of the point-to-point link. As with the Link Control Protocol, this is accomplished through an exchange of packets. IPCP packets may not be exchanged until LCP has reached the Network-Layer Protocol Configuration Negotiation phase. IPCP packets received before this phase is reached should be silently discarded. Likewise, IP datagrams may not be exchanged until IPCP has first opened the connection (reached the Open state).

The IP Control Protocol is exactly the same as the Link Control Protocol with the following exceptions:

Data Link Layer Protocol Field

Exactly one IP Control Protocol packet is encapsulated in the Information field of PPP Data Link Layer frames where the Protocol field indicates type hex 8021 (IP Control Protocol).

Code field

Only Codes 1 through 7 (Configure-Request, Configure-Ack, Configure-Nak, Configure-Reject, Terminate-Request, Terminate-Ack and Code-Reject) are used. Other Codes should be treated as unrecognized and should result in Code-Rejects.

Timeouts

IPCP packets may not be exchanged until the Link Control Protocol has reached the network-layer Protocol Configuration Negotiation phase. An implementation should be prepared to wait for Link Quality testing to finish before timing out waiting for a Configure-Ack or other response. It is suggested that an implementation give up only after user intervention or a configurable amount of time.

Configuration Option Types

The IPCP has a separate set of Configuration Options. The most up-to-date values of the type field are specified in the most recent "Assigned Numbers" RFC.

Sending IP Datagrams

Before any IP packets may be communicated, both the Link Control Protocol and the IP Control Protocol must reach the Open state.

Exactly one IP packet is encapsulated in the Information field of PPP Data Link Layer frames where the Protocol field indicates type hex 0021 (Internet Protocol).

The maximum length of an IP packet transmitted over a PPP link is the same as the maximum length of the Information field of a PPP data link layer frame. Larger IP datagrams must be fragmented as necessary. If a system wishes to avoid fragmentation and reassembly, it should use the TCP Maximum Segment Size option [RFC-879], or a similar mechanism, to discourage others from sending large datagrams.

Asynchronous HDLC

This appendix summarizes the modifications to ISO 3309-1979 proposed in ISO 3309:1984/PDAD1. These modifications allow HDLC to be used with 8-bit asynchronous links.

Transmission Considerations

Each octet is delimited by a start and a stop element.

Flag Sequence

The Flag Sequence is a single octet and indicates the beginning or end of a frame. The Flag Sequence consists of the binary sequence 01111110 (hexadecimal 0x7e).

Transparency

On asynchronous links, a character stuffing procedure is used. The Control Escape octet is defined as binary 01111101 (hexadecimal 0x7d) where the bit positions are numbered 87654321 (not 76543210, BEWARE).

After FCS computation, the transmitter examines the entire frame between the two Flag Sequences. Each Flag Sequence, Control Escape octet and octet with value less than hexadecimal 0x20 is replaced by a two octet sequence consisting of the Control Escape octet and the original octet with bit 6 complemented (i.e., exclusive-or'd with hexadecimal 0x20).

Prior to FCS computation, the receiver examines the entire frame between the two Flag Sequences. Each octet with value less than hexadecimal 0x20 is simply removed (it may have been inserted by intervening data communications equipment). For each Control Escape octet, that octet is also removed, but bit 6 of the following octet is complemented. A Control Escape octet immediately preceding the closing Flag Sequence indicates an invalid frame.

Note: The inclusion of all octets less than hexadecimal 0x20 allows all ASCII control characters excluding DEL (Delete) to be transparently communicated through almost all known data communications equipment.

A few examples may make this more clear. Packet data is transmitted on the link as follows:

```
0x7e is encoded as 0x7d, 0x5e.
0x7d is encoded as 0x7d, 0x5d.
0x01 is encoded as 0x7d, 0x21.
```

Aborting a Transmission

On asynchronous links, frames may be aborted by transmitting a "0" stop bit where a "1" bit is expected (framing error) or by transmitting a Control Escape octet followed immediately by a closing Flag Sequence.

Inter-frame Time Fill

On asynchronous links, inter-octet and inter-frame time fill should be accomplished by transmitting continuous "1" bits (mark-hold state).

Note: On asynchronous links, inter-frame time fill can be viewed as extended inter-octet time fill. Doing so can save one octet for every frame, decreasing delay and increasing bandwidth. This is possible since a Flag Sequence may serve as both a frame close and a frame begin. After having received any frame, an idle receiver will always be in a frame begin state.

Robust transmitters should avoid using this trick over-zealously since the price for decreased delay is decreased reliability. Noisy links may cause the receiver to receive garbage characters and interpret them as part of an incoming frame. If the transmitter does not transmit a new opening Flag Sequence before sending the next frame, then that frame will be appended to the noise characters causing an

invalid frame (with high reliability). Transmitters should avoid this by transmitting an open Flag Sequence whenever "appreciable time" has elapsed since the prior closing Flag Sequence. It is suggested that implementations will achieve the best results by always sending an opening Flag Sequence if the new frame is not back-to-back with the last. The maximum value for "appreciable time" is likely to be no greater than the typing rate of a slow to average typist, say 1 second.

Fast Frame Check Sequence (FCS) Implementation
FCS Computation Method

The following code provides a table lookup computation for calculating the Frame Check Sequence as data arrives at the interface. The table is created by the code in section 2.

```
/*
 * u16 represents an unsigned 16-bit number.  Adjust the typedef for
 * your hardware.
 */
typedef unsigned short u16;


/*
 * FCS lookup table as calculated by the table generator in section 2.
 */
static u16 fcstab[256] = {
   0x0000, 0x1189, 0x2312, 0x329b, 0x4624, 0x57ad, 0x6536, 0x74bf,
   0x8c48, 0x9dc1, 0xaf5a, 0xbed3, 0xca6c, 0xdbe5, 0xe97e, 0xf8f7,
   0x1081, 0x0108, 0x3393, 0x221a, 0x56a5, 0x472c, 0x75b7, 0x643e,
   0x9cc9, 0x8d40, 0xbfdb, 0xae52, 0xdaed, 0xcb64, 0xf9ff, 0xe876,
   0x2102, 0x308b, 0x0210, 0x1399, 0x6726, 0x76af, 0x4434, 0x55bd,
   0xad4a, 0xbcc3, 0x8e58, 0x9fd1, 0xeb6e, 0xfae7, 0xc87c, 0xd9f5,
   0x3183, 0x200a, 0x1291, 0x0318, 0x77a7, 0x662e, 0x54b5, 0x453c,
   0xbdcb, 0xac42, 0x9ed9, 0x8f50, 0xfbef, 0xea66, 0xd8fd, 0xc974,
   0x4204, 0x538d, 0x6116, 0x709f, 0x0420, 0x15a9, 0x2732, 0x36bb,
   0xce4c, 0xdfc5, 0xed5e, 0xfcd7, 0x8868, 0x99e1, 0xab7a, 0xbaf3,
   0x5285, 0x430c, 0x7197, 0x601e, 0x14a1, 0x0528, 0x37b3, 0x263a,
   0xdecd, 0xcf44, 0xfddf, 0xec56, 0x98e9, 0x8960, 0xbbfb, 0xaa72,
   0x6306, 0x728f, 0x4014, 0x519d, 0x2522, 0x34ab, 0x0630, 0x17b9,
   0xef4e, 0xfec7, 0xcc5c, 0xddd5, 0xa96a, 0xb8e3, 0x8a78, 0x9bf1,
   0x7387, 0x620e, 0x5095, 0x411c, 0x35a3, 0x242a, 0x16b1, 0x0738,
   0xffcf, 0xee46, 0xdcdd, 0xcd54, 0xb9eb, 0xa862, 0x9af9, 0x8b70,
   0x8408, 0x9581, 0xa71a, 0xb693, 0xc22c, 0xd3a5, 0xe13e, 0xf0b7,
   0x0840, 0x19c9, 0x2b52, 0x3adb, 0x4e64, 0x5fed, 0x6d76, 0x7cff,
   0x9489, 0x8500, 0xb79b, 0xa612, 0xd2ad, 0xc324, 0xf1bf, 0xe036,
   0x18c1, 0x0948, 0x3bd3, 0x2a5a, 0x5ee5, 0x4f6c, 0x7df7, 0x6c7e,
   0xa50a, 0xb483, 0x8618, 0x9791, 0xe32e, 0xf2a7, 0xc03c, 0xd1b5,
   0x2942, 0x38cb, 0x0a50, 0x1bd9, 0x6f66, 0x7eef, 0x4c74, 0x5dfd,
   0xb58b, 0xa402, 0x9699, 0x8710, 0xf3af, 0xe226, 0xd0bd, 0xc134,
   0x39c3, 0x284a, 0x1ad1, 0x0b58, 0x7fe7, 0x6e6e, 0x5cf5, 0x4d7c,
   0xc60c, 0xd785, 0xe51e, 0xf497, 0x8028, 0x91a1, 0xa33a, 0xb2b3,
   0x4a44, 0x5bcd, 0x6956, 0x78df, 0x0c60, 0x1de9, 0x2f72, 0x3efb,
   0xd68d, 0xc704, 0xf59f, 0xe416, 0x90a9, 0x8120, 0xb3bb, 0xa232,
   0x5ac5, 0x4b4c, 0x79d7, 0x685e, 0x1ce1, 0x0d68, 0x3ff3, 0x2e7a,
   0xe70e, 0xf687, 0xc41c, 0xd595, 0xa12a, 0xb0a3, 0x8238, 0x93b1,
   0x6b46, 0x7acf, 0x4854, 0x59dd, 0x2d62, 0x3ceb, 0x0e70, 0x1ff9,
   0xf78f, 0xe606, 0xd49d, 0xc514, 0xb1ab, 0xa022, 0x92b9, 0x8330,
   0x7bc7, 0x6a4e, 0x58d5, 0x495c, 0x3de3, 0x2c6a, 0x1ef1, 0x0f78
};

#define PPPINITFCS      0xffff  /* Initial FCS value */
#define PPPGOODFCS      0xf0b8  /* Good final FCS value */
```

```
/*
 * Calculate a new fcs given the current fcs and the new data.
 */
u16 pppfcs(fcs, cp, len)
    register u16 fcs;
    register unsigned char *cp;
    register int len;
{
    ASSERT(sizeof (u16) == 2);
    ASSERT(((u16) -1) > 0);
    while (len--)
        fcs = (fcs >> 8) ^ fcstab[(fcs ^ *cp++) & 0xff];

    return (fcs);
}
```

Fast Frame Check Sequence (FCS) Implementation
Fast FCS table generator

The following code creates the lookup table used to calculate the FCS.

```
/*
 * Generate a FCS table for the HDLC FCS.
 *
 * Drew D. Perkins at Carnegie Mellon University.
 *
 * Code liberally borrowed from Mohsen Banan and D. Hugh Redelmeier.
 */

/*
 * The HDLC polynomial: x**0 + x**5 + x**12 + x**16 (0x8408).
 */
#define P       0x8408


main()
{
    register unsigned int b, v;
    register int i;

    printf("typedef unsigned short u16;\n");
    printf("static u16 fcstab[256] = {");
    for (b = 0; ; ) {
        if (b % 8 == 0)
            printf("\n");

        v = b;
        for (i = 8; i--; )
            v = v & 1 ? (v >> 1) ^ P : v >> 1;

        printf("0x%04x", v & 0xFFFF);
        if (++b == 256)
            break;
        printf(",");
    }
    printf("\n};\n");
}
```

References

[1] Electronic Industries Association, EIA Standard RS-232-C, "Interface Between Data Terminal Equipment and Data Communications Equipment Employing Serial Binary Data Interchange", August 1969.

[2] International Organization For Standardization, ISO Standard 3309-1979, "Data communication - High-level data link control procedures - Frame structure", 1979.

[3] International Organization For Standardization, ISO Standard 4335-1979, "Data communication - High-level data link control procedures - Elements of procedures", 1979.

[4] International Organization For Standardization, ISO Standard 4335-1979/Addendum 1, "Data communication - High-level data link control procedures - Elements of procedures - Addendum 1", 1979.

[5] International Organization For Standardization, Proposed Draft International Standard ISO 3309:1983/PDAD1, "Information processing systems - Data communication - High-level data link control procedures - Frame structure - Addendum 1: Start/stop transmission", 1984.

[6] International Telecommunication Union, CCITT Recommendation X.25, "Interface Between Data Terminal Equipment (DTE) and Data Circuit Terminating Equipment (DCE) for Terminals Operating in the Packet Mode on Public Data Networks", CCITT Red Book, Volume VIII, Fascicle VIII.3, Rec. X.25., October 1984.

[7] Perez, "Byte-wise CRC Calculations", IEEE Micro, June, 1983.

[8] Morse, G., "Calculating CRC's by Bits and Bytes", Byte, September 1986.

[9] LeVan, J., "A Fast CRC", Byte, November 1987.

[10] American National Standards Institute, ANSI X3.4-1977, "American National Standard Code for Information Interchange", 1977.

[11] Postel, J., "Internet Protocol", RFC 791, USC/Information Sciences Institute, September 1981.

[12] Reynolds, J., and J. Postel, "Assigned Numbers", RFC 1060, USC/Information Sciences Institute, March 1990.

[13] Postel, J., "The TCP Maximum Segment Size Option and Related Topics", RFC 879, USC/Information Sciences Institute, November 1983.

Electronic Industries Association, EIA Standard RS-232-C, "Interface Between Data Terminal Equipment and Data Communications Equipment Employing Serial Binary Data Interchange", August 1969.

International Organization For Standardization, ISO Standard 3309-1979, "Data communication - High-level data link control procedures - Frame structure", 1979.

International Organization For Standardization, ISO Standard 4335-1979, "Data communication - High-level data link control procedures - Elements of procedures", 1979.

International Organization For Standardization, ISO Standard 4335-1979/Addendum 1, "Data communication - High-level data link control procedures - Elements of procedures - Addendum 1", 1979.

International Organization For Standardization, Proposed Draft International Standard ISO 3309:1983/PDAD1, "Information processing systems - Data communication - High-level data link control procedures - Frame structure - Addendum 1: Start/stop transmission", 1984.

International Telecommunication Union, CCITT Recommendation X.25, "Interface Between Data Terminal Equipment (DTE) and Data Circuit Terminating Equipment (DCE) for Terminals Operating in the Packet Mode on Public Data Networks", CCITT Red Book, Volume VIII, Fascicle VIII.3, Rec. X.25., October 1984.

Perez, "Byte-wise CRC Calculations", IEEE Micro, June, 1983.

Morse, G., "Calculating CRC's by Bits and Bytes", Byte, September 1986.

LeVan, J., "A Fast CRC", Byte, November 1987.

American National Standards Institute, ANSI X3.4-1977, "American National Standard Code for Information Interchange", 1977.

Chairman's Address

This proposal is the product of the Point-to-Point Protocol Working Group of the Internet Engineering Task Force (IETF). The working group can be contacted via the chair:

Russ Hobby
UC Davis
Computing Services
Davis, CA 95616

Phone: (916) 752-0236
EMail: rdhobby@ucdavis.edu

Author's Address

Questions about this memo can also be directed to the author:

Drew D. Perkins
Carnegie Mellon University
Networking and Communications
Pittsburgh, PA 15213

Phone: (412) 268-8576
EMail: ddp@andrew.cmu.edu

Acknowledgments

Many people spent significant time helping to develop the Point-to-Point Protocol. The complete list of people is too numerous to list, but the following people deserve special thanks: Ken Adelman (TGV), Craig Fox (NSC), Phill Gross (NRI), Russ Hobby (UC Davis), David Kaufman (Proteon), John LoVerso (Xylogics), Bill Melohn (Sun Microsystems), Mike Patton (MIT), Drew Perkins (CMU), Greg Satz (cisco systems) and Asher Waldfogel (Wellfleet).

RFC-1183 New DNS RR Definitions **(Updates: RFCs 1034, 1035)**

C. Everhart, Transarc
L. Mamakos, University of Maryland
R. Ullmann, Prime Computer
P. Mockapetris, Editor ISI
October 1990

This memo defines five new DNS types for experimental purposes. This RFC describes an Experimental Protocol for the Internet community, and requests discussion and suggestions for improvements. Distribution of this memo is unlimited.

Security issues are not addressed in this memo.

Introduction

This RFC defines the format of new Resource Records (RRs) for the Domain Name System (DNS), and reserves corresponding DNS type mnemonics and numerical codes. The definitions are in three independent sections: (1) location of AFS database servers, (2) location of responsible persons, and (3) representation of X.25 and ISDN addresses and route binding. All are experimental.

This RFC assumes that the reader is familiar with the DNS. The data shown is for pedagogical use and does not necessarily reflect the real Internet.

AFS Data Base Location

This section defines an extension of the DNS to locate servers both for AFS (AFS is a registered trademark of Transarc Corporation) and for the Open Software Foundation's (OSF) Distributed Computing Environment (DCE) authenticated naming system using HP/Apollo's NCA, both to be components of the OSF DCE. The discussion assumes that the reader is familiar with AFS and NCA.

The AFS (originally the Andrew File System) system uses the DNS to map from a domain name to the name of an AFS cell database server. The DCE Naming service uses the DNS for a similar function: mapping from the domain name of a cell to authenticated name servers for that cell. The method uses a new RR type with mnemonic AFSDB and type code of 18 (decimal).

AFSDB has the following format:

<owner> <ttl> <class> AFSDB <subtype> <hostname>

Both RDATA fields are required in all AFSDB RRs. The <subtype> field is a 16 bit integer. The <hostname> field is a domain name of a host that has a server for the cell named by the owner name of the RR.

The format of the AFSDB RR is class insensitive. AFSDB records cause type A additional section processing for <hostname>. This, in fact, is the rationale for using a new type code, rather than trying to build the same functionality with TXT RRs.

Note that the format of AFSDB in a master file is identical to MX. For purposes of the DNS itself, the subtype is merely an integer. The present subtype semantics are discussed below, but changes are possible and will be announced in subsequent RFCs.

In the case of subtype 1, the host has an AFS version 3.0 Volume Location Server for the named AFS cell. In the case of subtype 2, the host has an authenticated name server holding the cell-root directory node for the named DCE/NCA cell.

The use of subtypes is motivated by two considerations. First, the space of DNS RR types is limited. Second, the services provided are sufficiently distinct that it would continue to be confusing for a client to attempt to connect to a cell's servers using the protocol for one service, if the cell offered only the other service.

As an example of the use of this RR, suppose that the Toaster Corporation has deployed AFS 3.0 but not (yet) the OSF's DCE. Their cell, named toaster.com, has three "AFS 3.0 cell database server" machines: bigbird.toaster.com, ernie.toaster.com, and henson.toaster.com. These three machines would be listed in three AFSDB RRs. These might appear in a master file as:

```
toaster.com.    AFSDB   1 bigbird.toaster.com.
toaster.com.    AFSDB   1 ernie.toaster.com.
toaster.com.    AFSDB   1 henson.toaster.com.
```

As another example use of this RR, suppose that Femto College (domain name femto.edu) has deployed DCE, and that their DCE cell root directory is served by processes running on green.femto.edu and turquoise.femto.edu. Furthermore, their DCE file servers also run AFS

3.0-compatible volume location servers, on the hosts `turquoise.femto.edu` and `orange.femto.edu`. These machines would be listed in four AFSDB RRs, which might appear in a master file as:

```
femto.edu.   AFSDB   2 green.femto.edu.
femto.edu.   AFSDB   2 turquoise.femto.edu.
femto.edu.   AFSDB   1 turquoise.femto.edu.
femto.edu.   AFSDB   1 orange.femto.edu.
```

Responsible Person

The purpose of this section is to provide a standard method for associating responsible person identification to any name in the DNS.

The domain name system functions as a distributed database which contains many different form of information. For a particular name or host, you can discover it's Internet address, mail forwarding information, hardware type and operating system among others.

A key aspect of the DNS is that the tree-structured namespace can be divided into pieces, called zones, for purposes of distributing control and responsibility. The responsible person for zone database purposes is named in the SOA RR for that zone. This section describes an extension which allows different responsible persons to be specified for different names in a zone.

Identification of The Guilty Party

Often it is desirable to be able to identify the responsible entity for a particular host. When that host is down or malfunctioning, it is difficult to contact those parties which might resolve or repair the host. Mail sent to `POSTMASTER` may not reach the person in a timely fashion. If the host is one of a multitude of workstations, there may be no responsible person which can be contacted on that host.

The `POSTMASTER` mailbox on that host continues to be a good contact point for mail problems, and the zone contact in the SOA record for database problem, but the RP record allows us to associate a mailbox to entities that don't receive mail or are not directly connected (namespace-wise) to the problem (e.g., `GATEWAY.ISI.EDU` might want to point at `HOTLINE@BBN.COM`, and `GATEWAY` doesn't get mail, nor does the ISI zone administrator have a clue about fixing gateways).

RFC-1183 New DNS RR Definitions: Responsible Person

The Responsible Person RR

The method uses a new RR type with mnemonic RP and type code of 17 (decimal).

RP has the following format:

```
<owner> <ttl> <class> RP <mbox-dname> <txt-dname>
```

Both RDATA fields are required in all RP RRs.

The first field, <mbox-dname>, is a domain name that specifies the mailbox for the responsible person. Its format in master files uses the DNS convention for mailbox encoding, identical to that used for the RNAME mailbox field in the SOA RR. The root domain name (just ".") may be specified for <mbox-dname> to indicate that no mailbox is available.

The second field, <txt-dname>, is a domain name for which TXT RR's exist. A subsequent query can be performed to retrieve the associated TXT resource records at <txt-dname>. This provides a level of indirection so that the entity can be referred to from multiple places in the DNS. The root domain name (just ".") may be specified for <txt-dname> to indicate that the TXT_DNAME is absent, and no associated TXT RR exists.

The format of the RP RR is class insensitive. RP records cause no additional section processing. (TXT additional section processing for <txt-dname> is allowed as an option, but only if it is disabled for the root, i.e., ".").

The Responsible Person RR can be associated with any node in the Domain Name System hierarchy, not just at the leaves of the tree.

The TXT RR associated with the TXT_DNAME contain free format text suitable for humans.

Multiple RP records at a single name may be present in the database. They should have identical TTLs.

Responsible Person RR EXAMPLES

Some examples of how the RP record might be used.

```
sayshell.umd.edu. A     128.8.1.14
                  MX    10 sayshell.umd.edu.
                  HINFO NeXT UNIX
                  WKS   128.8.1.14 tcp ftp telnet smtp
                  RP    louie.trantor.umd.edu.  LAM1.people.umd.edu.

LAM1.people.umd.edu. TXT (
      "Louis A. Mamakos, (301) 454-2946, don't call me at home!" )
```

In this example, the responsible person's mailbox for the host `SAYSHELL.UMD.EDU` is `louie@trantor.umd.edu`. The TXT RR at `LAM1.people.umd.edu` provides additional information and advice.

```
TERP.UMD.EDU.     A     128.8.10.90
                  MX    10 128.8.10.90
                  HINFO MICROVAX-II UNIX
                  WKS   128.8.10.90 udp domain
                  WKS   128.8.10.90 tcp ftp telnet smtp domain
                  RP    louie.trantor.umd.edu. LAM1.people.umd.edu.
                  RP    root.terp.umd.edu. ops.CS.UMD.EDU.

TRANTOR.UMD.EDU. A     128.8.10.14
                  MX    10 trantor.umd.edu.
                  HINFO MICROVAX-II UNIX
                  WKS   128.8.10.14 udp domain
                  WKS   128.8.10.14 tcp ftp telnet smtp domain
                  RP    louie.trantor.umd.edu. LAM1.people.umd.edu.
                  RP    petry.netwolf.umd.edu. petry.people.UMD.EDU.
                  RP    root.trantor.umd.edu. ops.CS.UMD.EDU.
                  RP    gregh.sunset.umd.edu.  .

LAM1.people.umd.edu.  TXT   "Louis A. Mamakos (301) 454-2946"
petry.people.umd.edu. TXT   "Michael G. Petry (301) 454-2946"
ops.CS.UMD.EDU.       TXT   "CS Operations Staff (301) 454-2943"
```

This set of resource records has two hosts, `TRANTOR.UMD.EDU` and `TERP.UMD.EDU`, as well as a number of TXT RRs. Note that `TERP.UMD.EDU` and `TRANTOR.UMD.EDU` both reference the same pair of TXT resource records, although the mail box names (`root.terp.umd.edu` and `root.trantor.umd.edu`) differ.

Here, we obviously care much more if the machine flakes out, as we've specified four persons which might want to be notified of problems or other events involving `TRANTOR.UMD.EDU`. In this example, the last RP RR for `TRANTOR.UMD.EDU` specifies a mailbox (`gregh.sunset.umd.edu`), but no associated TXT RR.

X.25 and ISDN Addresses, Route Binding

This section describes an experimental representation of X.25 and ISDN addresses in the DNS, as well as a route binding method, analogous to the MX for mail routing, for very large scale networks.

There are several possible uses, all experimental at this time. First, the RRs provide simple documentation of the correct addresses to use in static configurations of IP/X.25 and SMTP/X.25.

The RRs could also be used automatically by an internet network-layer router, typically IP. The procedure would be to map IP address to domain name, then name to canonical name if needed, then following RT records, and finally attempting an IP/X.25 call to the address found. Alternately, configured domain names could be resolved to identify IP to X.25/ISDN bindings for a static but periodically refreshed routing table.

This provides a function similar to ARP for wide area non-broadcast networks that will scale well to a network with hundreds of millions of hosts.

Also, a standard address binding reference will facilitate other experiments in the use of X.25 and ISDN, especially in serious inter-operability testing. The majority of work in such a test is establishing the n-squared entries in static tables.

Finally, the RRs are intended for use in a proposal by one of the authors for a possible next-generation internet.

The X25 RR

The X25 RR is defined with mnemonic X25 and type code 19 (decimal).

X25 has the following format:

```
<owner> <ttl> <class> X25 <PSDN-address>
```

<PSDN-address> is required in all X25 RRs.

<PSDN-address> identifies the PSDN (Public Switched Data Network) address in the X.121 numbering plan associated with <owner>. Its format in master files is a <character-string> syntactically identical to that used in TXT and HINFO.

The format of X25 is class insensitive. X25 RRs cause no additional section processing.

The <PSDN-address> is a string of decimal digits, beginning with the 4 digit DNIC (Data Network Identification Code), as specified in X.121. National prefixes (such as a 0) MUST NOT be used.

For example:

```
Relay.Prime.COM.  X25       311061700956
```

The ISDN RR

The ISDN RR is defined with mnemonic ISDN and type code 20 (decimal).

An ISDN (Integrated Service Digital Network) number is simply a telephone number. The intent of the members of the CCITT is to upgrade all telephone and data network service to a common service.

The numbering plan (E.163/E.164) is the same as the familiar international plan for POTS (an un-official acronym, meaning Plain Old Telephone Service). In E.166, CCITT says "An E.163/E.164 telephony subscriber may become an ISDN subscriber without a number change."

ISDN has the following format:

```
<owner> <ttl> <class> ISDN <ISDN-address> <sa>
```

The <ISDN-address> field is required; <sa> is optional.

<ISDN-address> identifies the ISDN number of <owner> and DDI (Direct Dial In) if any, as defined by E.164 and E.163, the ISDN and PSTN (Public Switched Telephone Network) numbering plan. E.163 defines the country codes, and E.164 the form of the addresses. Its format in master files is a <character-string> syntactically identical to that used in TXT and HINFO.

<sa> specifies the subaddress (SA). The format of <sa> in master files is a <character-string> syntactically identical to that used in TXT and HINFO.

The format of ISDN is class insensitive. ISDN RRs cause no additional section processing.

The <ISDN-address> is a string of characters, normally decimal digits, beginning with the E.163 country code and ending with the DDI if any. Note that ISDN, in Q.931, permits any IA5 character in the general case.

The <sa> is a string of hexadecimal digits. For digits 0-9, the concrete encoding in the Q.931 call setup information element is identical to BCD.

For example:

```
Relay.Prime.COM.    IN   ISDN      150862028003217
sh.Prime.COM.       IN   ISDN      150862028003217 004
```

(Note: "1" is the country code for the North American Integrated Numbering Area, i.e., the system of "area codes" familiar to people in those countries.)

The RR data is the ASCII representation of the digits. It is encoded as one or two <character-string>s, i.e., count followed by characters.

CCITT recommendation E.166 defines prefix escape codes for the representation of ISDN (E.163/E.1641183_REF_9) addresses in X.121, and PSDN (X.121) addresses in E.164. It specifies that the exact codes are a "national matter", i.e., different on different networks. A host connected to the ISDN may be able to use both the X25 and ISDN addresses, with the

local prefix added.

The Route Through RR

The Route Through RR is defined with mnemonic RT and type code 21 (decimal).

The RT resource record provides a route-through binding for hosts that do not have their own direct wide area network addresses. It is used in much the same way as the MX RR .

RT has the following format:

```
<owner> <ttl> <class> RT <preference> <intermediate-host>
```

Both RDATA fields are required in all RT RRs.

The first field, <preference>, is a 16 bit integer, representing the preference of the route. Smaller numbers indicate more preferred routes.

<intermediate-host> is the domain name of a host which will serve as an intermediate in reaching the host specified by <owner>. The DNS RRs associated with <intermediate-host> are expected to include at least one A, X25 ISDN record.

The format of the RT RR is class insensitive. RT records cause type X25, ISDN, and A additional section processing for <intermediate- host>.

For example,

```
sh.prime.com.       IN   RT   2    Relay.Prime.COM.
                    IN   RT   10   NET.Prime.COM.
*.prime.com.        IN   RT   90   Relay.Prime.COM.
```

When a host is looking up DNS records to attempt to route a datagram, it first looks for RT records for the destination host, which point to hosts with address records (A, X25, ISDN) compatible with the wide area networks available to the host. If it is itself in the set of RT records, it discards any RTs with preferences higher or equal to its own. If there are no (remaining) RTs, it can then use address records of the destination itself.

Wild-card RTs are used exactly as are wild-card MXs. RT's do not "chain"; that is, it is not valid to use the RT RRs found for a host referred to by an RT.

The concrete encoding is identical to the MX RR.

Mockapetris, P., "Domain Names - Concepts and Facilities", RFC 1034, USC/Information Sciences Institute, November 1987.

Mockapetris, P., "Domain Names - Implementation and Specification", RFC 1035, USC/Information Sciences Institute, November 1987.

Spector A., and M. Kazar, "Uniting File Systems", UNIX Review,
7(3), pp. 61-69, March 1989.

Zahn, et al., "Network Computing Architecture", Prentice-Hall, 1989.

International Telegraph and Telephone Consultative Committee, "Numbering Plan for the International Telephone Service", CCITT Recommendations E.163., IXth Plenary Assembly, Melbourne, 1988, Fascicle II.2 ("Blue Book").

International Telegraph and Telephone Consultative Committee, "Numbering Plan for the ISDN Era", CCITT Recommendations E.164., IXth Plenary Assembly, Melbourne, 1988, Fascicle II.2 ("Blue Book").

International Telegraph and Telephone Consultative Committee.
"Numbering Plan Interworking in the ISDN Era", CCITT Recommendations E.166., IXth Plenary Assembly, Melbourne, 1988, Fascicle II.2 ("Blue Book").

International Telegraph and Telephone Consultative Committee, "International Numbering Plan for the Public Data Networks", CCITT Recommendations X.121., IXth Plenary Assembly, Melbourne, 1988, Fascicle VIII.3 ("Blue Book"); provisional, Geneva, 1978; amended, Geneva, 1980, Malaga-Torremolinos, 1984 and Melborne, 1988.

Korb, J., "Standard for the Transmission of IP datagrams Over Public Data Networks", RFC 877, Purdue University, September 1983.

Ullmann, R., "SMTP on X.25", RFC 1090, Prime Computer, February 1989.

Ullmann, R., "TP/IX: The Next Internet", Prime Computer
(unpublished), July 1990.

Authors' Addresses

Craig F. Everhart
Transarc Corporation
The Gulf Tower
707 Grant Street
Pittsburgh, PA 15219

Phone: +1 412 338 4467

EMail: Craig_Everhart@transarc.com

Louis A. Mamakos
Network Infrastructure Group
Computer Science Center
University of Maryland
College Park, MD 20742-2411

Phone: +1-301-405-7836

Email: louie@Sayshell.UMD.EDU

Robert Ullmann 10-30
Prime Computer, Inc.
500 Old Connecticut Path
Framingham, MA 01701

Phone: +1 508 620 2800 ext 1736

Email: Ariel@Relay.Prime.COM

Paul Mockapetris
USC Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292

Phone: 213-822-1511

EMail: pvm@isi.edu

–

RFC-1189 The Common Management Information Services and Protocol for the Internet (CMOT & CMIT)

U. Warrier, L. Besaw, L. LaBarre, B. Handspicker

Status of this Memo

This memo defines a network management architecture that uses the International Organization for Standardization's (ISO) Common Management Information Services/Common Management Information Protocol (CMIS/CMIP) in the Internet. This RFC specifies an IAB standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "IAB Official Protocol Standards" for the standardization state and status of this protocol.

Distribution of this memo is unlimited.

RFC-1189 Common Management Information Services

Overview

This memo is a revision of RFC-1095 - "The Common Management Information Services and Protocol over TCP/IP". It defines a network management architecture that uses the International Organization for Standardization's (ISO) Common Management Information Services/Common Management Information Protocol (CMIS/CMIP) in the Internet. This architecture provides a means by which control and monitoring information can be exchanged between a manager and a remote network element. In particular, this memo defines the means for implementing the International Standard (IS) version of CMIS/CMIP on top of both IP-based and OSI-based Internet transport protocols for the purpose of carrying management information defined in the Internet-standard management information base. Together with the relevant ISO standards and the companion RFCs that describe the initial structure of management information and management information base, these documents provide the basis for a comprehensive architecture and system for managing both IP- based and OSI-based internets, and in particular the Internet.

In creating this revision of RFC-1095, the following technical and editorial changes were made:

1) The tutorial section on OSI Management included in RFC-1095 has been removed from this document. After some revisions, the tutorial material may be published as another RFC.
2) The sections in RFC-1095 which discussed the semantics of how to interpret requests in the context of Internet MIBs has been removed from this protocol document. This topic is now discussed in the OIM-MIB-II draft document. This protocol should be useable with MIB-I or MIB-II. But, it will also be able to exploit the new features of the OIM-MIB-II.
3) This document is based on the final International Standards for CMIS/CMIP (ISO 9595/9596) rather than the Draft International Standards.
4) Many of the original agreements defined in RFC-1095 have been accepted and included in the OIW NMSIG implementers agreements. Rather than duplicating these agreements, they have been removed from this memo. This document should be read in conjunction with ISO 9595/9596 (CMIS/CMIP) and the OIW Stable Agreements document.
5) The Association Negotiation describe in RFC-1095 has been changed to align with current international and national agreements. But, it has retained backwards compatibility with the assignment of an Application Context Name which is identical to the Application Context Name specified in RFC-1095.

RFC-1189 Common Management Information Services

Introduction

This memo is the output of the OSI Internet Management Working Group of the Internet Engineering Task Force (IETF). As directed by the Internet Activites Board (IAB) in RFC-1052, it addresses the need for a long-term network management system based on ISO CMIS/CMIP. This memo contains a set of protocol agreements for implementing a network management system based on these ISO Management standards. Now that CMIS/CMIP has been voted an International Standard (IS), it has become a stable basis for product development. This profile specifies how to apply CMIP to management of both IP-based and OSI- based Internet networks. Network management using ISO CMIP to manage IP-based networks will be refered to as "CMIP Over TCP/IP" (CMOT). Network management using ISO CMIP to manage OSI-based networks will be refered to as "CMIP". This memo specifies the protocol agreements necessary to implement CMIP and accompanying ISO protocols over OSI, TCP and UDP transport protocols.

This memo must be read in conjunction with ISO and Internet documents defining specific protocol standards. Documents defining the following ISO standards are required for the implementor: Abstract Syntax Notation One (ASN.1) [5, 6], Association Control (ACSE) [7, 8], Remote Operations (ROSE) [9, 10], Common Management Information Services (CMIS) [11] and Common Management Information Protocol (CMIP) [12] with their addenda [32-35]. The specification of a lightweight presentation layer protocol[RFC-1085] is required for use with the CMOT section of this profile. The SMI [RFC-1155]), the MIB-I [RFC-1066), the MIB-II [RFC-1156), and the OIM-MIB-II (see [29]) are used with this management system.

This memo is divided into sections for each of the protocols for which implementors' agreements are needed: CMISE, ACSE, ROSE, and, for CMOT, the lightweight presentation protocol. The protocol profile defined in this memo draws on the technical work of the OSI Network Management Forum [14] and the Network Management Special Interest Group (NMSIG) of the National Institute of Standards and Technology (NIST) (formerly the National Bureau of Standards) [30]. Wherever possible, an attempt has been made to either directly reference or remain consistent with the protocol agreements reached by these groups.

Protocol Overview

This part of the document is a specification of the protocols of the OIM architecture. Contained herein are the agreements required to implement interoperable network management systems using these protocols. The protocol suite defined by these implementors' agreements will facilitate communication between equipment of different vendors, suppliers, and networks. This will allow the emergence of powerful multivendor network management based on ISO models and protocols.

The choice of a set of protocol standards together with further agreements needed to implement those standards is commonly referred to as a "profile." The selection policy for this profile is to use existing standards from the international standards community (ISO and CCITT) and the Internet community. Existing ISO standards and draft standards in the area of OSI network management form the basis of this profile. Other ISO application layer standards (ROSE and ACSE) are used to support the ISO management protocol (CMIP). To ensure interoperability, certain choices and restrictions are made here concerning various options and parameters provided by these standards. Internet standards are used to provide the underlying network transport. These agreements provide a precise statement of the implementation choices made for implementing ISO network management standards in IP-based and OSI-based internets.

In addition to the OIM working group, there are at least two other bodies actively engaged in defining profiles for interoperable OSI network management: the OSI Implementors Workshop (OIW) and the OSI Network Management Forum. Both of these groups are similar to the OIM working group in that they are each defining profiles for using ISO standards for network management. Both differ in that they are specifying the use only of underlying ISO protocols, while the OIM working group is concerned with using OSI management in both OSI and TCP/IP networks. In the interest of greater future compatibility, the OIM working group has attempted to make this profile conform as closely as possible to the ongoing work of these two bodies.

This section will describe the CMOT Protocol Suite, the CMIP Protocol Suite and Conformance Requirements common to both CMOT and CMIP. Later sections will specify the implementers agreements for specific layer protocols that comprise the CMOT and CMIP Protocol Suites.

RFC-1189 Common Management Information Services - Protocol Overview

The CMOT Protocol Suite

The following seven protocols compose the CMOT protocol suite: ISO ACSE, ISO DIS ROSE, ISO CMIP, the lightweight presentation protocol (LPP), UDP, TCP, and IP. The relation of these protocols to each other is briefly summarized in Figure 2.

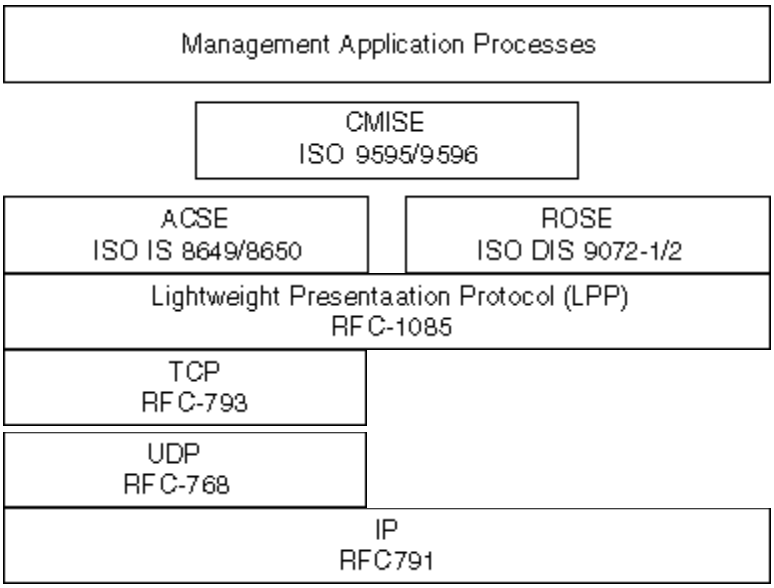


Figure 2. The CMOT Protocol Suite

The CMIP Protocol Suite

The following six protocols compose the CMIP protocol suite: ISO ACSE, ISO DIS ROSE, ISO CMIP, ISO Presentation, ISO Session and ISO Transport. The relation of these protocols to each other is briefly summarized in Figure 3.

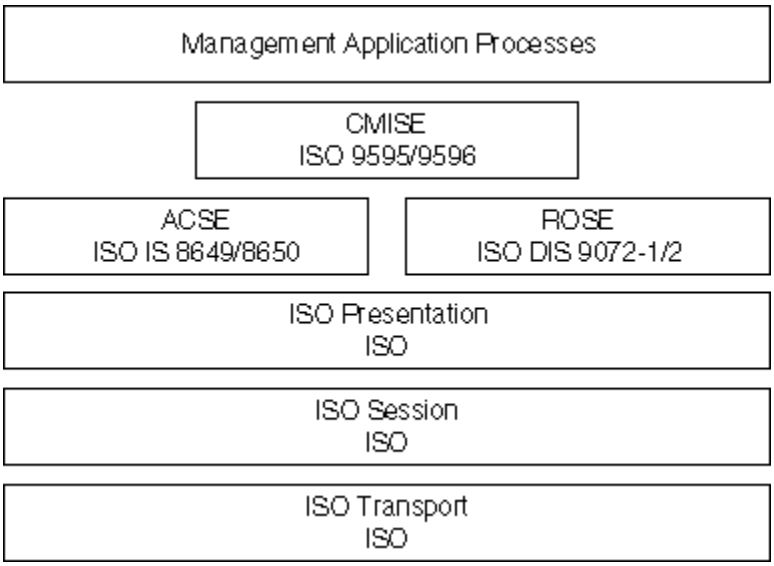


Figure 3. The CMIP Protocol Suite

Conformance Requirements

A CMOT-conformant system must implement the following protocols: ACSE, ROSE, CMIP, LPP, and IP. A CMOT-conformant system must support the use of the LPP over either UDP or TCP. The use of the LPP over both UDP and TCP on the same system may be supported.

A CMIP-conformant system must implement the following protocols: ACSE, ROSE, CMIP, ISO Presentation, ISO Session and ISO Transport.

Common Management Information Service Element

The Common Management Information Service Element (CMISE) is specified in two ISO documents. The service definition for the Common Management Information Service (CMIS) is given in ISO 9595 [11]. The protocol specification for the Common Management Information Protocol (CMIP) is found in ISO 9596 [12]. In addition, the addenda for add/remove support in M-SET [32, 34] must be supported for both CMOT and CMIP. The addenda for M-CANCEL-GET [33, 35] may be supported by an implementation, but it's use is negotiated as part of association negotiation.

Association Policies

The following ACSE services are required by CMISE: A-ASSOCIATE, A- RELEASE, A-ABORT, and A-P-ABORT. The rest of the CMIP protocol uses the RO-INVOKE, RO-RESULT, RO-ERROR, and RO-REJECT services of ROSE.

There are four types of association that may be negotiated between managing and managed systems. These types are:

Event	M-EVENT-REPORTs may be sent by the managed system; no other CMIP PDUs are allowed
Event/Monitor	same as Event type except that, in addition, the managing system may also issue M-GET requests and receive M-GET responses over the association
Monitor/Control	managing system may issue M-GET, M-SET, M-CREATE, M-DELETE and M-ACTION requests over the association; no event reporting is allowed
Full Mgr/Agent	all functions must be supported

A conformant system must support at least one of these Association types. Note that a system may play both managing and managed system roles, but not on the same association.

The negotiation process uses the A-ASSOCIATE and A-RELEASE services. Application Context Name is used to determine the requestor's "role" in an association (as managing or managed system) and to determine the type of the association.

The following values for Application Context Name are registered for for CMOT and CMIP:

{iso(1) identified-organization(3) dod(6)
internet(1) mgmt(2) mib(1) oim(9) acn(1)
cmot1095(1)}
(for backwards compatible negotiation with RFC 1095 CMOT implementations)

{iso(1) identified-organization(3) dod(6)
internet(1) mgmt(2) mib(1) oim(9) acn(1)
manager-event-association(2)}

{iso(1) identified-organization(3) dod(6)
internet(1) mgmt(2) mib(1) oim(9) acn(1)
manager-event-monitor-association(3)}

{iso(1) identified-organization(3) dod(6)
internet(1) mgmt(2) mib(1) oim(9) acn(1)
manager-monitor-control-association(4)}

{iso(1) identified-organization(3) dod(6)
internet(1) mgmt(2) mib(1) oim(9) acn(1)
manager-full-association(5)}

{iso(1) identified-organization(3) dod(6)
internet(1) mgmt(2) mib(1) oim(9) acn(1)
agent-event-association(6)}

The following negotiation rules are to be used:

1. A managed system may only request an Event association and, in fact, must create an Event association if it has an event to report and no suitable association already exists.
2. Managing systems may request any association type.
3. An association is created by the requesting system issuing an A-ASSOCIATE request with the requestor's AE-TITLE and the desired application context. The responding system then returns either 1) an A-ASSOCIATE response with the requestor's AE-TITLE and the application context which it wishes to accept or 2) an A-ASSOCIATE response rejecting the association.
4. Managed systems may negotiate "downward" from Full to Monitor/Control, Event/Monitor or Event by returning the new application context in the A-ASSOCIATE response to the managing system during the association creation process. In the same fashion, managed systems may negotiate from Event/Monitor to Event.
5. When a managing system receives an application context in an A-ASSOCIATE response that differs from the context sent in an A-ASSOCIATE request it may either proceed with the new context or refuse the new context by issuing an A-RELEASE request.

A-RELEASE is used when the requestor does not agree with the new context. A-ABORT is used for invalid negotiation. If A-ABORT were to be used to terminate an association, there exists the potential for loss of information, such as pending events or confirmations. A-ABORT must be used, however, when a protocol violation occurs or where an association is not yet established.

General Agreements on Users of CMIS

The general agreements on users of CMIS shall be as specified in the OIW Stable Agreements [30] section 18.6.2.

The following additional agreements are specified.

- A system need only implement the services and service primitives required for the association types that it supports.
- Current/Event times shall be fields shall use 1 millisecond granularity. If the system generating the PDU does not have the current time, yet does have the time since last boot, then GeneralizedTime can be used to encode this information. The time since last boot will be added to the base time "0001 Jan 1 00:00:00.00" using the Gregorian calendar algorithm. (In the Gregorian calendar, all years have 365 days except those divisible by 4 and not by 400, which have 366.) The use of the year 1 as the base year will prevent any confusion with current time.

  If no meaningful time is available, then the year 0 shall be used in GeneralizedTime to indicate this fact.

Specific Agreements on Users of CMIS

The specific agreements on users of CMIS shall be as specified in the OIW Stable Agreements [30] section 18.6.3.

The following additional agreements are specified:

- Event time shall be mandatory for all events.
- Both the "managed Object Class" and "managed Object Instance" parameters must be present in the following CMIS Service Response/Confirmation primitives: the M-EVENT-REPORT Confirmed, the M-GET, the M-SET, the M-ACTION, the M-CREATE, and the M-DELETE.

CMIP Agreements

The CMIS and CMIP implementers agreements documented in the OIW Stable Implementers Agreements [30] plus those mandated by the CMIP standard will be used for both CMOT and CMIP. In addition to these implementers agreements, the following specific agreements must be observed:

o An implementation is required to support all filter items except subsetOf, supersetOf, nonNullSetIntersection, and substrings.

o The "managedObjectInstance" field must be present in the ProcessingFailure Error PDU. The "managedObjectClass" field must be present in the NoSuchArgument Error PDU.

[Temporary Note: The CMIS/P implementers agreements have reach a fairly stable status in the OIW working agreements document. It is expected that the CMIS/P agreements (18.6.2 and 18.6.3) will be recommended to be moved into the stable agreements document during either the June 1990 meetings. Reference [30] points to the presumed June 1990 updated version of the stable agreements document.]

Services Required by CMIP

The services required by CMIP shall be as specified in the OIW Stable Implementors Agreements [30] section 18.6.5.

The following additional agreements are specified:

- ASCE Requirements: Application contexts shall be as defined in section 4.1 of these agreements. The values and defaults of parameters to the ACSE parameters given to the presentation service are specified in RFC 1085 [13] for CMOT and in the NIST Stable Implementers Agreements [30] for CMIP.
- Presentation Requirements: CMOT implementations shall be supported by the Lightweight Presentation Protocol (LPP) [13]. The LPP may use either TCP or UDP. When UDP is used, an implementation need not accept LPP PDUs whose length exceeds 484 octets.
- Session Requirements: CMOT implementations will not require the session protocol.

Acknowledgements

This RFC is the result of the work of many people. The following members of the IETF OSI Internet Management and preceding Netman working groups made important contributions:

Amatzia Ben-Artzi, Synoptics
Asheem Chandna, AT&T Bell Laboratories
Ken Chapman, Digital Equipment Corporation
Anthony Chung, Sytek
George Cohn, Ungermann-Bass
Gabriele Cressman, Sun Microsystems
Tom Halcin, Hewlett-Packard
Pranati Kapadia, Hewlett-Packard
Lee LaBarre, The MITRE Corporation (co-chair)
Dave Mackie, 3Com
Keith McCloghrie, Hughes/InterLan
Jim Robertson, 3Com
Milt Roselinsky, CMC
Marshall Rose, PSI
John Scott, Data General
Lou Steinberg, IBM

Authors' Addresses

Unnikrishnan S. Warrier
NetLabs
11693 San Vicente Blvd
Suite 348
Los Angeles, CA 90049

Phone: (213) 476-4070
Email: unni@netlabs.com

Larry Besaw
Hewlett-Packard
3404 East Harmony Road
Fort Collins, CO 80525

Phone: (303) 229-6022
Email: lmb%hpcndaw@hplabs.hp.com

Lee LaBarre
Mitre
Burlington Road
Bedford, MA 01730

Phone: (617) 271-8507
Email: cel@mbunix.mitre.org

Brian D. Handspicker
Digital Equipment Corporation
550 King St.
Littleton, Ma. 01460

Phone: (508) 486-7894
Email: bd@vines.enet.dec.com

7. References

[1] Cerf, V., "IAB Recommendations for the Development of Internet Network Management Standards", RFC 1052, IAB, April 1988.

[2] Rose, M., and K. McCloghrie, "Structure and Identification of Management Information for TCP/IP-based internets", RFC 1065, TWG, August 1988.

[3] McCloghrie, K., and M. Rose, "Management Information Base for Network Management of TCP/IP-based internets", RFC 1066, TWG, August 1988.

[4] Case, J., M. Fedor, M. Schoffstall, and J. Davin, "A Simple Network Management Protocol (SNMP)", RFC 1098, (Obsoletes RFC 1067), University of Tennessee at Knoxville, NYSERNet, Inc., Rensselaer Polytechnic Institute, MIT Laboratory for Computer Science, April 1989.

[5] ISO 8824: "Information Processing Systems - Open Systems Interconnection, Specification of Abstract Syntax Notation One (ASN.1)", Geneva, March 1988.

[6] ISO 8825: "Information Processing Systems - Open Systems Interconnection, Specification of Basic Encoding Rules for Abstract Notation One (ASN.1)", Geneva, March 1988.

[7] ISO 8649: "Information Processing Systems - Open Systems Interconnection, Service Definition for Association Control Service Element".

[8] ISO 8650: "Information Processing Systems - Open Systems Interconnection, Protocol Specification for Association Control Service Element".

[9] CCITT Recommendation X.219, Working Document for ISO 9072-1: "Information processing systems - Text Communication, Remote

Operations: Model, Notation and Service Definition", Gloucester, November 1987.

[10] CCITT Recommendation X.229, Working Document for ISO 9072-2: "Information processing systems - Text Communication, Remote Operations: Protocol Specification", Gloucester, November 1987.

[11] ISO 9595: "Information Processing Systems - Open Systems Interconnection, Management Information Service Definition - Part 2: Common Management Information Service", 22 December 1988.

[12] ISO 9596: "Information Processing Systems - Open Systems Interconnection, Management Information Protocol Specification - Part 2: Common Management Information Protocol", 22 December 1988.

[13] Rose, M., "ISO Presentation Services on top of TCP/IP-based internets", RFC 1085, TWG, December 1988.

[14] OSI Network Management Forum, "Forum Interoperable Interface Protocols", September 1988.

[15] ISO DIS 7498-4: "Information Processing Systems - Open Systems Interconnection, Basic Reference Model - Part 4: OSI Management Framework".

[16] ISO/IEC JTC1/SC21/WG4 N571: "Information Processing Systems - Open Systems Interconnection, Systems Management: Overview", London, July 1988.

[17] Klerer, S. Mark, "The OSI Management Architecture: An Overview", IEEE Network Magazine, March 1988.

[18] Ben-Artzi, A., "Network Management for TCP/IP Networks: An Overview", Internet Engineering Task Force working note, April 1988.

[19] ISO/IEC JTC1/SC21/WG4 N3324: "Information Processing Pystems - Open Systems Interconnection, Management Information Services - Structure of Management Information - Part I: Management Information Model", Sydney, December 1988.

[20] Postel, J., "User Datagram Protocol", RFC 768, USC/Information Sciences Institute, August 1980.

[21] Postel, J., "Transmission Control Protocol", RFC 793, USC/Information Sciences Institute, September 1981.

[22] ISO DP 9534: "Information processing systems - Open Systems Interconnection, Application Layer Structure", 10 March 1987.

[23] Rose, M., and D. Cass, "ISO Transport Services on top of the TCP, Version: 3", RFC 1006, Northrop Research and Technology Center, May 1987.

[24] ISO 8822: "Information Processing Systems - Open Systems Interconnection, Connection Oriented Presentation Service Definition", June 1987.

[25] Postel, J., "Internet Protocol", RFC 791, USC/Information Sciences Institute, September 1981.

[26] CCITT Draft Recommendation X.500, ISO 9594/1-8: "The Directory", Geneva, March 1988.

[27] Warrier, U. and L. Besaw, "The Common Management Information Services and Protocol over TCP/IP (CMOT)", RFC 1095, Unisys Corporation, Hewlett-Packard, April 1989.

[28] McCloghrie, K., and M. Rose, "Management Information Base for Network Management of TCP/IP-based internets", RFC 1156, Hughes LAN Systems, Performance Systems International, May 1990.

[29] LaBarre, L., "OIM MIB-II", working note, December 1989.

[30] NIST NMSIG, "NIST Stable Implementers Agreements", NIST Special Publication 500-162, as ammended by June 1990.

[31] NIST NMSIG, "NIST Working Implementers Agreements", December 1989.

[32] ISO IS 9595 1989: DAD1: "CMIS Add/Remove Addendum".

[33] ISO IS 9595 1989: DAD2: "CMIS Cancel-Get Addendum".

[34] ISO IS 9596 1989: DAD1: "CMIP Add/Remove Addendum".

[35] ISO IS 9596 1989: DAD2: "CMIP Cancel-Get Addendum".

8. Security Considerations

Security issues are not discussed in this memo.

RFC-1196 The Finger User Information Protocol

D. Zimmerman
Center for Discrete Mathematics
and
Threoretical Computer Science
Rutgers University
December 1990

Status of this Memo

This memo defines a protocol for the exchange of user information. This RFC specifies an IAB standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "IAB Official Protocol Standards" for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Abstract

This memo describes the Finger User Information Protocol. This is a simple protocol which provides an interface to a remote user information program.

Based on RFC-742, a description of the original Finger protocol, this memo attempts to clarify the expected communication between the two ends of a Finger connection. It also tries not to invalidate the many existing implementations or add unnecessary restrictions to the original protocol definition. This edition corrects and clarifies in a minor way, RFC-1194.

RFC-1196 Finger Protocol

Introduction

RFC-1196 Finger Protocol - Introduction

Intent

This memo describes the Finger User Information Protocol. This is a simple protocol which provides an interface to a remote user information program (RUIP).

Based on RFC-742, a description of the original Finger protocol, this memo attempts to clarify the expected communication between the two ends of a Finger connection. It also tries not to invalidate the many current implementations or add unnecessary restrictions to the original protocol definition.

The most prevalent implementations of Finger today seem to be primarily derived from the BSD UNIX work at the University of California, Berkeley. Thus, this memo is based around the BSD version's behavior.

However, the BSD version provides few options to tailor the Finger RUIP for a particular site's security policy, or to protect the user from dangerous data. Furthermore, there are MANY potential security holes that implementors and administrators need to be aware of, particularly since the purpose of this protocol is to return information about a system's users, a sensitive issue at best. Therefore, this memo makes a number of important security comments and recommendations.

History

The FINGER program at SAIL, written by Les Earnest, was the inspiration for the NAME program on ITS. Earl Killian at MIT and Brian Harvey at SAIL were jointly responsible for implementing the original protocol.

Ken Harrenstien is the author of RFC-742, "Name/Finger", which this memo began life as.

Requirements

In this document, the words that are used to define the significance of each particular requirement are capitalized. These words are:

* "**must**"
  This word or the adjective "REQUIRED" means that the item is an absolute requirement of the specification.
* "**should**"
  This word or the adjective "RECOMMENDED" means that there may exist valid reasons in particular circumstances to ignore this item, but the full implications should be understood and the case carefully weighed before choosing a different course.
* "**may**"
  This word or the adjective "OPTIONAL" means that this item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because it enhances the product, for example; another vendor may omit the same item.

An implementation is not compliant if it fails to satisfy one or more of the **must** requirements. An implementation that satisfies all the **must** and all the **should** requirements is said to be "unconditionally compliant"; one that satisfies all the **must** requirements but not all the **should** requirements is said to be "conditionally compliant".

Use of the Protocol

RFC-1196 Finger Protocol - Use of the Protocol

Flow of Events

Finger is based on the Transmission Control Protocol, using TCP port 79 decimal (117 octal). A TCP connection is opened to a remote host on the Finger port. An RUIP becomes available on the remote end of the connection to process the request. The RUIP is sent a one line query based upon the Finger query specification. The RUIP processes the query, returns an answer, then closes the connection normally.

Data Format

Any data transferred **must** be in ASCII format, with no parity, and with lines ending in CRLF (ASCII 13 followed by ASCII 10). This excludes other character formats such as EBCDIC, etc. This also means that any characters between ASCII 128 and ASCII 255 should truly be international data, not 7-bit ASCII with the parity bit set.

Query Specifications

An RUIP **must** accept the entire Finger query specification.

The Finger query specification is defined:

```
{Q1}   ::= [{U}] [/W] {C}
{Q2}   ::= [{U}]{H} [/W] {C}
{U}    ::= username
{H}    ::= @hostname | @hostname{H}
{C}    ::= <CRLF>
```

{H}, being recursive, means that there is no arbitrary limit on the number of @hostname tokens in the query. In examples of the {Q2} request specification, the number of @hostname tokens is limited to two, simply for brevity.

Be aware that {Q1} and {Q2} do not refer to a user typing "finger user@host" from an operating system prompt. It refers to the line that an RUIP actually receives. So, if a user types "finger user@host<CRLF>", the RUIP on the remote host receives "user<CRLF>", which corresponds to {Q1}.

As with anything in the IP protocol suite, "be liberal in what you accept".

RUIP {Q2} Behavior

A query of {Q2} is a request to forward a query to another RUIP. An RUIP **must** either provide or actively refuse this forwarding service (see section 3.2.1). If an RUIP provides this service, it **must** conform to the following behavior:

Given that:

Host <H1> opens a Finger connection <F1-2> to an RUIP on host <H2>.

<H1> gives the <H2> RUIP a query <Q1-2> of type {Q2} (e.g., FOO@HOST1@HOST2).

It should be derived that:

Host <H3> is the right-most host in <Q1-2> (i.e., HOST2)

Query <Q2-3> is the remainder of <Q1-2> after removing the right-most "@hostname" token in the query (i.e., FOO@HOST1)

And so:

The <H2> RUIP then must itself open a Finger connection <F2-3> to <H3>, using <Q2-3>.

The <H2> RUIP must return any information received from <F2-3> to <H1> via <F1-2>.

The <H2> RUIP must close <F1-2> in normal circumstances only when the <H3> RUIP closes <F2-3>.

Expected RUIP Response

For the most part, the output of an RUIP doesn't follow a strict specification, since it is designed to be read by people instead of programs. It should mainly strive to be informative.

Output of ANY query is subject to the discussion in the security section.

RFC-1196 Finger Protocol - Expected RUIP Response

{C} Query

A query of {C} is a request for a list of all online users. An RUIP **must** either answer or actively refuse (see section 3.2.2). If it answers, then it **must** provide at least the user's full name. The system administrator **should** be allowed to include other useful information (per section 3.2.3), such as:

- terminal location
- office location
- office phone number
- job name
- idle time (number of minutes since last typed input, or since last job activity).

RFC-1196 Finger Protocol - Expected RUIP Response

{U}{C} Query

A query of {U}{C} is a request for in-depth status of a specified user {U}. If you really want to refuse this service, you probably don't want to be running Finger in the first place.

An answer **must** include at least the full name of the user. If the user is logged in, at least the same amount of information returned by {C} for that user **must** also be returned by {U}{C}.

Since this is a query for information on a specific user, the system administrator **should** be allowed to choose to return additional useful information (per section 3.2.3), such as:

- office location
- office phone number
- home phone number
- status of login (not logged in, logout time, etc)
- user information file

A user information file is a feature wherein a user may leave a short message that will be included in the response to Finger requests. (This is sometimes called a "plan" file.) This is easily implemented by (for example) having the program look for a specially named text file in the user's home directory or some common area; the exact method is left to the implementor. The system administrator **should** be allowed to specifically turn this feature on and off. See section 3.2.4 for caveats.

There **may** be a way for the user to run a program in response to a Finger query. If this feature exists, the system administrator **should** be allowed to specifically turn it on and off. See the section on "Execution of User Programs" for caveats.

{U} Ambiguity

Allowable "names" in the command line **must** include "user names" or "login names" as defined by the system. If a name is ambiguous, the system administrator **should** be allowed to choose whether or not all possible derivations should be returned in some fashion (per section 3.2.6).

RFC-1196 Finger Protocol - Expected RUIP Response

/W Query Token

The token /W in the {Q1} or {Q2} query types **should** at best be interpreted at the last RUIP to signify a higher level of verbosity in the user information output, or at worst be ignored.

RFC-1196 Finger Protocol - Expected RUIP Response

Vending Machines

Vending machines **should** respond to a {C} request with a list of all items currently available for purchase and possible consumption. Vending machines **should** respond to a {U}{C} request with a detailed count or list of the particular product or product slot. Vending machines should NEVER NEVER EVER eat requests. Or money.

Security

Implementation Security

Sound implementation of Finger is of the utmost importance. Implementations should be tested against various forms of attack. In particular, an RUIP **should** protect itself against malformed inputs. Vendors providing Finger with the operating system or network software should subject their implementations to penetration testing.

Finger is one of the avenues for direct penetration, as the Morris worm pointed out quite vividly. Like Telnet, FTP and SMTP, Finger is one of the protocols at the security perimeter of a host. Accordingly, the soundness of the implementation is paramount. The implementation should receive just as much security scrutiny during design, implementation, and testing as Telnet, FTP, or SMTP.

RUIP Security

Warning!! Finger discloses information about users; moreover, such information may be considered sensitive. Security administrators should make explicit decisions about whether to run Finger and what information should be provided in responses. One existing implementation provides the time the user last logged in, the time he last read mail, whether unread mail was waiting for him, and who the most recent unread mail was from! This makes it possible to track conversations in progress and see where someone's attention was focused. Sites that are information-security conscious should not run Finger without an explicit understanding of how much information it is giving away.

{Q2} Refusal

For individual site security concerns, the system administrator **should** be given an option to individually turn on or off RUIP processing of {Q2}. If RUIP processing of {Q2} is turned off, the RUIP **must** return a service refusal message of some sort. "Finger forwarding service denied" is adequate. The purpose of this is to allow individual hosts to choose to not forward Finger requests, but if they do choose to, to do so consistently.

Overall, there are few cases which would warrant processing of {Q2} at all, and they are far outweighed by the number of cases for refusing to process {Q2}. In particular, be aware that if a machine is part of security perimeter (that is, it is a gateway from the outside world to some set of interior machines), then turning {Q2} on provides a path through that security perimeter. Therefore, it is RECOMMENDED that the default of the {Q2} processing option be to refuse processing. It certainly should not be enabled in gateway machines without careful consideration of the security implications.

{C} Refusal

For individual site security concerns, the system administrator **should** be given an option to individually turn on or off RUIP acceptance of {C}. If RUIP processing of {C} is turned off, the RUIP **must** return a service refusal message of some sort. "Finger online user list denied" is adequate. The purpose of this is to allow individual hosts to choose to not list the users currently online.

Atomic Discharge

All implementations of Finger **should** allow individual system administrators to tailor what atoms of information are returned to a query. For example:

- Administrator A should be allowed to specifically choose to return office location, office phone number, home phone number, and logged in/logout time.
- Administrator B should be allowed to specifically choose to return only office location, and office phone number.
- Administrator C should be allowed to specifically choose to return the minimum amount of required information, which is the person's full name.

User Information Files

Allowing an RUIP to return information out of a user-modifiable file should be seen as equivalent to allowing any information about your system to be freely distributed. That is, it is potentially the same as turning on all specifiable options. This information security breach can be done in a number of ways, some cleverly, others straightforwardly. This should disturb the sleep of system administrators who wish to control the returned information.

Execution of User Programs

Allowing an RUIP to run a user program in response to a Finger query is potentially dangerous. BE CAREFUL!! -- the RUIP **must not** allow system security to be compromised by that program. Implementing this feature may be more trouble than it is worth, since there are always bugs in operating systems, which could be exploited via this type of mechanism.

{U} Ambiguity

Be aware that a malicious user's clever and/or persistent use of this feature can result in a list of most of the usernames on a system. Refusal of {U} ambiguity should be considered in the same vein as refusal of {C} requests.

Audit Trails

Implementations **should** allow system administrators to log Finger queries.

Client Security

It is expected that there will normally be some client program that the user runs to query the initial RUIP. By default, this program **should** filter any unprintable data, leaving only printable 7-bit characters (ASCII 32 through ASCII 126), tabs (ASCII 9), and CRLFs. This is to protect against people playing with terminal escape codes, changing other peoples' X window names, or committing other dastardly or confusing deeds. Two separate user options **should** be considered to modify this behavior, so that users may choose to view international or control characters:

- one to allow all characters less than ASCII 32
- another to allow all characters greater than ASCII 126

For environments that live and breathe international data, the system administrator **should** be given a mechanism to enable the latter option by default for all users on a particular system. This can be done via a global environment variable or similar mechanism.

Examples

Example with a Null Command Line ({C})

Site: elbereth.rutgers.edu

Command line: <CRLF>

```
Login       Name               TTY Idle    When            Office
rinehart Mark J. Rinehart      p0  1:11 Mon 12:15  019 Hill       x3166
greenfie Stephen J. Greenfiel  p1       Mon 15:46  542 Hill       x3074
rapatel  Rocky - Rakesh Patel  p3    4d Thu 00:58  028 Hill       x2287
pleasant Mel Pleasant          p4    3d Thu 21:32  019 Hill    908-932-
dphillip Dave Phillips         p5  021: Sun 18:24  265 Hill       x3792
dmk      David Katinsky        p6    2d Thu 14:11  028 Hill       x2492
cherniss Cary Cherniss         p7     5 Mon 15:42  127 Psychol    x2008
harnaga  Doug Harnaga          p8  2:01 Mon 10:15  055 Hill       x2351
brisco   Thomas P. Brisco      pe  2:09 Mon 13:37  h055           x2351
laidlaw  Angus Laidlaw         q0  1:55 Mon 11:26  E313C       648-5592
cje      Chris Jarocha-Ernst   q1     8 Mon 13:43  259 Hill       x2413
```

Example with Name Specified ({U}{C})

Site: dimacs.rutgers.edu

Command line: pirmann<CRLF>

```
Login name: pirmann                     In real life: David Pirmann
Office: 016 Hill, x2443                 Home phone: 989-8482
Directory: /dimacs/u1/pirmann           Shell: /bin/tcsh
Last login Sat Jun 23 10:47 on ttyp0 from romulus.rutgers.
No unread mail
Project:
Plan:
                    Work Schedule, Summer 1990
                Rutgers LCSR Operations, 908-932-2443

                   Monday       5pm - 12am
                   Tuesday      5pm - 12am
                   Wednesday    9am -  5pm
                   Thursday     9am -  5pm
                   Saturday     9am -  5pm

                      larf larf hoo hoo
```

RFC-1196 Finger Protocol - Examples

Example with Ambiguous Name Specified ({U}{C})

```
Site: elbereth.rutgers.edu
Command line: ron<CRLF>
Login name: spinner                     In real life: Ron Spinner
Office: Ops Cubby,    x2443             Home phone: 463-7358
Directory: /u1/spinner                  Shell: /bin/tcsh
Last login Mon May  7 16:38 on ttyq7
Plan:
            ught i
          ca      n
        m           a
       '      ...     t
      I      .   .     i
             !         m
      !       !       e
       p       !pool
        l
         e
          H



Login name: surak                       In real life: Ron Surak
Office: 000 OMB Dou,    x9256
Directory: /u2/surak                    Shell: /bin/tcsh
Last login Fri Jul 27 09:55 on ttyq3
No Plan.

Login name: etter                       In real life: Ron Etter
Directory: /u2/etter                    Shell: /bin/tcsh
Never logged in.
No Plan.
```

Example of Query Type {Q2} ({U}{H}{H}{C})

Site: dimacs.rutgers.edu

Command line: hedrick@math.rutgers.edu@pilot.njin.net<CRLF>

```
[pilot.njin.net]
[math.rutgers.edu]
Login name: hedrick                     In real life: Charles Hedrick
Office: 484 Hill, x3088
Directory: /math/u2/hedrick             Shell: /bin/tcsh
Last login Sun Jun 24 00:08 on ttyp1 from monster-gw.rutge
No unread mail
No Plan.
```

Acknowledgments

Thanks to everyone in the Internet Engineering Task Force for their comments. Special thanks to Steve Crocker for his security recommendations and prose.

Author's Address

David Paul Zimmerman
Center for Discrete Mathematics and
Theoretical Computer Science
Rutgers University
P.O. Box 1179
Piscataway, NJ 08855-1179

Phone: (908)932-4592

EMail: dpz@dimacs.rutgers.edu

RFC-1206 FYI on Questions and Answers to Commonly asked "New Internet User" Questions

G. Malkin; FTP Software, Inc.
A. Marine; SRI
February 1991
FYI #4

Status of this Memo

This FYI RFC is one of two FYI's called, "Questions and Answers" (Q/A), produced by the User Services Working Group of the Internet Engineering Task Force (IETF). The goal is to document the most commonly asked questions and answers in the Internet.

This memo provides information for the Internet community. It does not specify any standard. Distribution of this memo is unlimited.

RFC-1206 FYI on Questions and Answers for "New Internet Users"

Introduction

New users joining the Internet community have the same questions as did everyone else who has ever joined. Our quest is to provide the Internet community with up to date, basic Internet knowledge and experience, while moving the redundancies away from the electronic mailing lists so that the lists' subscribers do not have to read the same queries and answers over and over again.

Future updates of this memo will be produced as User Services members become aware of additional questions that should be included, and of deficiencies or inaccuracies that should be amended in this document. An additional FYI Q/A will be published which will deal with intermediate and advanced Q/A topics.

The Q/A mailing lists are maintained by Gary Malkin at FTP.COM. They are used by a subgroup of the User Services Working Group to discuss the Q/A FYIs. They include:

quail@ftp.com	This is a discussion mailing list. Its primary use is for pre-release review of the Q/A FYIs.
quail-request@ftp.com	This is how you join the quail mailing list.
quail-box@ftp.com	This is a write-only list which serves as a repository for candidate questions and answers. It is not necessary to be on the quail mailing list to forward to the quail-box.

Acknowledgements

The following people deserve thanks for their help and contributions to this FYI Q/A: Vint Cerf (CNRI), Ralph Droms (Bucknell), Tracy LaQuey Parker (UTexas), Craig Partridge (SICS), Jon Postel (ISI), Joyce K. Reynolds (ISI), Karen Roubicek (BBNST), Marty Schoffstall (PSI, Inc.), Patricia Smith (Merit), Gene Spafford (Purdue) and James Van Bokkelen (FTP Software, Inc.).

RFC-1206 FYI on Questions and Answers for "New Internet Users"

Questions About the Internet

What is the Internet?

The Internet is a large collection of networks (all of which run the TCP/IP protocols) that are tied together so that users of any of the networks can use the network services provided by TCP/IP to reach users on any of the other networks. The Internet started with the ARPANET, but now includes such networks as NSFNET, NYSERnet, and thousands of others. There are other major wide area networks, such as BITNET and DECnet networks, that are not based on the TCP/IP protocols and are thus not part of the Internet. However, it is possible to communicate between them and the Internet via electronic mail because of mail gateways that act as "translators" between the different network protocols involved.

Note: You will often see "internet" with a small "i". This could refer to any network built based on TCP/IP, or might refer to networks using other protocol families that are composites built of smaller networks.

I just got on the Internet. What can I do now?

You now have access to all the resources you are authorized to use on your own Internet host, on any other Internet host on which you have an account, and on any other Internet host that offers publicly accessible information. The Internet gives you the ability to move information between these hosts via file transfers. Once you are logged into one host, you can use the Internet to open a connection to another, login, and use its services interactively (this is known as remote login or "TELNETTING". In addition, you can send electronic mail to users at any Internet site and to users on many non-Internet sites that are accessible via electronic mail.

There are various other services you can use. For example, some hosts provide access to specialized databases or to archives of information. The Internet Resource Guide provides information regarding some of these sites. The Internet Resource Guide lists facilities on the Internet that are available to users. Such facilities include supercomputer centers, library catalogs and specialized data collections. The guide is published by the NSF Network Service Center (NNSC) and is continuously being updated. The Resource Guide is distributed free via e-mail (send a note to resource-guide-request@nnsc.nsf.net to join the e-mail distribution) and via anonymous FTP (in nnsc.nsf.net:resource- guide/*). Hardcopy is available at a nominal fee (to cover reproduction costs) from the NNSC. Call the NNSC at 617-873-3400 for more information.

How do I find out if a site has a computer on the Internet?

Three good sources to consult are "!%@:: A Directory of Electronic Mail Addressing and Networks" by Donnalyn Frey and Rick Adams; "The User's Directory of Computer Networks", by Tracy LaQuey; and "The Matrix: Computer Networks and Conferencing Systems Worldwide", by John Quarterman.

In addition, it is possible to find some information about Internet sites in the WHOIS database maintained at the DDN NIC at SRI International. The DDN NIC (Defense Data Network, Network Information Center) provides an information retrieval interface to the database that is also called WHOIS. To use this interface, TELNET to NIC.DDN.MIL and type "whois" (carriage return). No login is necessary. Type "help" at the whois prompt for more information on using the facility. WHOIS will show many sites, but may not show every site registered with the DDN NIC (simply for reasons having to do with how the program is set up to search the database).

RFC-1206 FYI on Questions and Answers for "New Internet Users"

Questions About TCP/IP

What is TCP/IP?

TCP/IP (Transmission Control Protocol/Internet Protocol) [4,5,6] is the common name for a family of over 100 data-communications protocols used to organize computers and data-communications equipment into computer networks. TCP/IP was developed to interconnect hosts on ARPANET, PRNET (packet radio), and SATNET (packet satellite). All three of these networks have since been retired; but TCP/IP lives on. It is currently used on a large international network of networks called the Internet, whose members include universities, other research institutions, government facilities, and many corporations. TCP/IP is also sometimes used for other networks, particularly local area networks that tie together numerous different kinds of computers or tie together engineering workstations.

What are the other well-known standard protocols in the TCP/IP family?

Other than TCP and IP, the three main protocols in the TCP/IP suite are the Simple Mail Transfer Protocol (SMTP) [RFC-821], the File Transfer Protocol (FTP) [RFC-959], and the TELNET Protocol [RFC-854]. There are many other protocols in use on the Internet. The Internet Activities Board (IAB) regularly publishes an RFC [RFC-1250] under the title "IAB Official Protocol Standards" that describes the state of standardization of the various Internet protocols. This document is the best guide to the current status of Internet protocols and their recommended usage.

RFC-1206 FYI on Questions and Answers for "New Internet Users"

Questions About the Domain Name System

What is the Domain Name System?

The Domain Name System (DNS) is a hierarchical, distributed method of organizing the name space of the Internet. The DNS administratively groups hosts into a hierarchy of authority that allows addressing and other information to be widely distributed and maintained. A big advantage to the DNS is that using it eliminates dependence on a centrally-maintained file that maps host names to addresses.

What is a Fully Qualified Domain Name?

A Fully Qualified Domain Name (FQDN) is a domain name that includes all higher level domains relevant to the entity named. If you think of the DNS as a tree-structure with each node having its own label, a Fully Qualified Domain Name for a specific node would be its label followed by the labels of all the other nodes between it and the root of the tree. For example, for a host, a FQDN would include the string that identifies the particular host, plus all domains of which the host is a part up to and including the top-level domain (the root domain is always null). For example, `PARIS.NISC.SRI.COM` is a Fully Qualified Domain Name for the host at 192.33.33.109. In addition, `NISC.SRI.COM` is the FQDN for the NISC domain.

RFC-1206 FYI on Questions and Answers for "New Internet Users"

Questions About Internet Documentation

What is an RFC?

The Request for Comments documents (RFCs) are working notes of the Internet research and development community. A document in this series may be on essentially any topic related to computer communication, and may be anything from a meeting report to the specification of a standard. Submissions for Requests for Comments may be sent to the RFC Editor, Jon Postel (POSTEL@ISI.EDU).

Most RFCs are the descriptions of network protocols or services,often giving detailed procedures and formats for their implementation. Other RFCs report on the results of policy studies or summarize the work of technical committees or workshops. All RFCs are considered public domain unless explicitly marked otherwise.

While RFCs are not refereed publications, they do receive technical review from either the task forces, individual technical experts, or the RFC Editor, as appropriate. Currently, most standards are published as RFCs, but not all RFCs specify standards.

Anyone can submit a document for publication as an RFC. Submissions must be made via electronic mail to the RFC Editor. Please consult RFC 1111, "Instructions to RFC Authors" [RFC-1111], for further information. RFCs are accessible online in public access files, and a short message is sent to a notification distribution list indicating the availability of the memo. Requests to be added to this distribution list should be sent to RFC-REQUEST@NIC.DDN.MIL.

The online files are copied by interested people and printed or displayed at their sites on their equipment. (An RFC may also be returned via electronic mail in response to an electronic mail query.) This means that the format of the online files must meet the constraints of a wide variety of printing and display equipment.

Once a document is assigned an RFC number and published, that RFC is never revised or re-issued with the same number. There is never a question of having the most recent version of a particular RFC. However, a protocol (such as File Transfer Protocol (FTP)) may be improved and re-documented many times in several different RFCs. It is important to verify that you have the most recent RFC on a particular protocol. The "IAB Official Protocol Standards" [RFC-1250] memo is the reference for determining the correct RFC to refer to for the current specification of each protocol.

How do I obtain RFCs?

RFCs can be obtained via FTP from NIC.DDN.MIL, with the pathname RFC:RFCnnnn.TXT or RFC:RFCnnnn.PS (where "nnnn" refers to the number of the RFC). Login using FTP, username "anonymous" and password "guest". The NIC also provides an automatic mail service for those sites which cannot use FTP. Address the request to SERVICE@NIC.DDN.MIL and in the subject field of the message indicate the RFC number, as in "Subject: RFC nnnn" (or "Subject: RFC nnnn.PS" for PostScript RFCs).

RFCs can also be obtained via FTP from NIS.NSF.NET. Using FTP, login with username "anonymous" and password "guest"; then connect to the RFC directory ("cd RFC"). The file name is of the form RFCnnnn.TXT-1 (where "nnnn" refers to the number of the RFC). The NIS also provides an automatic mail service for those sites which cannot use FTP. Address the request to NIS-INFO@NIS.NSF.NET and leave the subject field of the message blank. The first line of the text of the message must be "SEND RFCnnnn.TXT-1", where nnnn is replaced by the RFC number.

Requests for special distribution should be addressed to either the author of the RFC in question, or to NIC@NIC.DDN.MIL. SRI International operates NIC.DDN.MIL and has a hardcopy subscription service for RFCs as well as several publications which incorporate a selection of RFCs defining Internet standards. Unless specifically noted otherwise on the RFC itself, all RFCs are for unlimited distribution.

How do I obtain a list of RFCs?

The NIC maintains a file that is an index of the RFCs. It lists each RFC, starting with the most recent, and for each RFC provides the number, title, author(s), issue date, and number of hardcopy pages. In addition, it lists the online formats (PostScript or ASCII text) for each RFC and the number of bytes each such version is online on the `NIC.DDN.MIL` host. If an RFC is also an FYI, that fact is noted, with the corresponding FYI number. (There is a parallel FYI Index available). Finally, the Index notes whether or not an RFC is obsoleted or updated by another RFC, and gives the number of that RFC, or if an RFC itself obsoletes or updates another RFC, and gives that RFC number. The index is updated online each time an RFC is issued.

This RFC Index is available online from the `NIC.DDN.MIL` host as `RFC:RFC-INDEX.TXT`. The FYI Index is online as `FYI:FYI-INDEX.TXT`. It is also available from the NIC in hardcopy for $10, as are individual RFCs. Call the NIC at 1-800-235-3155 for help in obtaining the file.

Which RFCs are Standards?

See "IAB Official Protocol Standards" [RFC-1250].

What is an Internet Draft? Are there any guidelines available for writing one?

Internet Drafts (I-D's) are the current working documents of the IETF. Internet Drafts are generally in the format of an RFC with some key differences:

- The Internet Drafts are not RFC's and are not a numbered document series.
- The words INTERNET-DRAFT appear in place of RFC XXXX in the upper left-hand corner.
- The document does not refer to itself as an RFC or as a Draft RFC.
- An Internet Draft does not state nor imply that it is a proposed standard. To do so conflicts with the role of the IAB, the RFC Editor, and the Internet Engineering Steering Group (IESG).

An Internet Drafts Directory has been installed to make available, for review and comment by the IETF members, draft documents that will be submitted ultimately to the IAB and the RFC Editor to be considered for publishing as an RFC. The Internet Drafts Directories are maintained primarily at the NSFNET Network Service Center (NNSC). There are several "shadow" machines which contain the IETF and Internet Drafts Directories. They are:

NSF Network Service Center: nnsc.nsf.net
DDN NIC: nic.ddn.mil
Pacific Rim: munnari.oz.au
Europe: nic.nordu.net (192.36.148.17)

To access these directories, use anonymous FTP. Login with username, "anonymous", password, "guest". Once logged in, change to the directory, "cd internet-drafts". Internet Draft files can then be retrieved.

For further information on the Internet Drafts of the IETF, or if you have problems with retrieving Internet Draft documents, contact Megan Davies (`mdavies@nri.reston.va.us`) or Greg Vaudreuil (`gvaudre@nri.reston.va.us`) for assistance.

How do I obtain OSI Standards documents?

OSI Standards documents are NOT available from the Internet via anonymous FTP due to copyright restrictions. These are available from:

Omnicom Information Service
501 Church Street NE
Suite 304
Vienna, VA 22180 USA
Telephone: (800) 666-4266 or (703) 281-1135
Fax: (703) 281-1505

However, the GOSIP specification which covers the use of OSI protocols within the U.S. Government is available from the NIC and from the National Institute of Standards and Technology (NIST). The final text of GOSIP Version 2 is now available from both sites. Version 2 is expected to become a Federal Information Processing Standard (FIPS) in early 1991.

Online sources:

Available through anonymous ftp from osi.ncsl.nist.gov (129.6.48.100) as:

```
./pub/gosip/gosip_v2.txt        -- ascii
./pub/gosip/gosip_v2.txt.Z      -- ascii compressed
./pub/gosip/gosip_v2.ps         -- PostScript
./pub/gosip/gosip_v2.ps.Z       -- PostScript compressed
```

Available through anonymous ftp from nic.ddn.mil (192.67.67.20) as:

```
PROTOCOLS:GOSIP-V2.TXT        -- ascii
PROTOCOLS:GOSIP-V2.PS         -- PostScript
```

Hardcopy sources:

Standards Processing Coordinator (ADP)
National Institute of Standards and Technology
Technology Building, Room B-64
Gaithersburg, MD 20899
(301) 975-2816

Network Information Systems Center
SRI International, Room EJ291
333 Ravenswood Ave.
Menlo Park, CA 94025
1-800-235-3155

RFC-1206 FYI on Questions and Answers for "New Internet Users"

Questions about Internet Organizations and Contacts

What is the IAB?

The Internet Activities Board (IAB) is the coordinating committee for Internet design, engineering and management [RFC-1160]. IAB members are deeply committed to making the Internet function effectively and evolve to meet a large scale, high speed future. The chairman serves a term of two years and is elected by the members of the IAB. The current Chair of the IAB is Vint Cerf. The IAB focuses on the TCP/IP protocol suite, and extensions to the Internet system to support multiple protocol suites.

The IAB performs the following functions:

1) Sets Internet Standards,
2) Manages the RFC publication process,
3) Reviews the operation of the IETF and IRTF,
4) Performs strategic planning for the Internet, identifying long-range problems and opportunities,
5) Acts as an international technical policy liaison and representative for the Internet community, and
6) Resolves technical issues which cannot be treated within the IETF or IRTF frameworks.

The IAB has two principal subsidiary task forces:

1) Internet Engineering Task Force (IETF)
2) Internet Research Task Force (IRTF)

Each of these Task Forces is led by a chairman and guided by a Steering Group which reports to the IAB through its chairman. For the most part, a collection of Research or Working Groups carries out the work program of each Task Force.

All decisions of the IAB are made public. The principal vehicle by which IAB decisions are propagated to the parties interested in the Internet and its TCP/IP protocol suite is the Request for Comments (RFC) note series and the Internet Monthly Report.

What is the IANA?

The task of coordinating the assignment of values to the parameters of protocols is delegated by the Internet Activities Board (IAB) to the Internet Assigned Numbers Authority (IANA). These protocol parameters include op-codes, type fields, terminal types, system names, object identifiers, and so on. The "Assigned Numbers" Request for Comments (RFC) [RFC-1060] documents the currently assigned values from several series of numbers used in network protocol implementations. Internet addresses and Autonomous System numbers are assigned by the Network Information Center at SRI International. This responsibility has been delegated by the IANA to the DDN NIC which serves as the Internet Registry. The IANA is located at USC/Information Sciences Institute.

Current types of assignments listed in Assigned Numbers and maintained by the IANA are:

Address Resolution Protocol Parameters
ARPANET and MILNET X.25 Address Mappings
ARPANET and MILNET Logical Addresses
ARPANET and MILNET Link Numbers
BOOTP Parameters and BOOTP Extension Codes
Domain System Parameters
IANA Ethernet Address Blocks
Ethernet Numbers of Interest
IEEE 802 Numbers of Interest
Internet Protocol Numbers
Internet Version Numbers
IP Time to Live Parameter
IP TOS Parameters
Machine Names
Mail Encryption Types
Multicast Addresses
Network Management Parameters
Point-to-Point Protocol Field Assignments
PRONET 80 Type Numbers
Port Assignments
Protocol and Service Names
Protocol/Type Field Assignments
Public Data Network Numbers
Reverse Address Resolution Protocol Operation Codes
TELNET Options
Terminal Type Names
Unix Ports
X.25 Type Numbers

For more information on number assignments, contact `IANA@ISI.EDU`.

What is a NIC? What is a NOC?

"NIC" stands for Network Information Center. It is an organization which provides network users with information about services provided by the network.

"NOC" stands Network Operations Center. It is an organization that is responsible for maintaining a network.

For many networks, especially smaller, local networks, the functions of the NIC and NOC are combined. For larger networks, such as mid-level and backbone networks, the NIC and NOC organizations are separate, yet they do need to interact to fully perform their functions.

What is "The NIC"?

"The NIC" is the Defense Data Network, Network Information Center (DDN NIC) at SRI International, which is a network information center which holds a primary repository for RFCs and Internet Drafts. The host name is NIC.DDN.MIL. Shadow copies of the RFCs and the Internet Drafts are maintained by the NSFNET on NIS.NSF.NET.

The DDN NIC also provides various user assistance services for DDN users; contact `NIC@NIC.DDN.MIL` or call 1-800-235-3155 for more information. In addition, the DDN NIC is the Internet registration authority for the root domain and several top and second level domains; maintains the official DoD Internet Host Table; is the site of the Internet Registry (IR); and maintains the WHOIS database of network users, hosts, domains, networks, and Points of Contact.

What is the IR?

The Internet Registry (IR) is the organization that is responsible for assigning identifiers, such as IP network numbers and autonomous system numbers, to networks. The IR also gathers and registers such assigned information. The IR may, in the future, allocate the authority to assign network identifiers to other organizations; however, it will continue to gather data regarding such assignments. At present, the DDN NIC at SRI International serves as the IR.

What is the IETF?

The Internet has grown to encompass a large number of widely geographically dispersed networks in academic and research communities. It now provides an infrastructure for a broad community with various interests. Moreover, the family of Internet protocols and system components has moved from experimental to commercial development. To help coordinate the operation, management and evolution of the Internet, the IAB established the Internet Engineering Task Force (IETF).

The IETF is chaired by Phill Gross and managed by its Internet Engineering Steering Group (IESG). The IETF is a large open community of network designers, operators, vendors, and researchers concerned with the Internet and the Internet protocol suite. It is organized around a set of several technical areas, each managed by a technical area director. In addition to the IETF Chairman, the area directors make up the IESG membership.

The IAB has delegated to the IESG the general responsibility for making the Internet work and for the resolution of all short- and mid-range protocol and architectural issues required to make the Internet function effectively.

What is the IRTF?

To promote research in networking and the development of new technology, the IAB established the Internet Research Task Force (IRTF).

In the area of network protocols, the distinction between research and engineering is not always clear, so there will sometimes be overlap between activities of the IETF and the IRTF. There is, in fact, considerable overlap in membership between the two groups. This overlap is regarded as vital for cross-fertilization and technology transfer.

The IRTF is a community of network researchers, generally with an Internet focus. The work of the IRTF is governed by its Internet Research Steering Group (IRSG). The chairman of the IRTF and IRSG is David Clark.

RFC-1206 FYI on Questions and Answers for "New Internet Users"

Questions About Services

How do I find someone's electronic mail address?

There are a number of directories on the Internet; however, all of them are far from complete. The largest directories are the WHOIS database at the DDN NIC, the PSInet White Pages, and KNOWBOT. Generally, it is still necessary to ask the person for his or her email address.

How do I use the WHOIS program at the DDN NIC?

To use the WHOIS program to search the WHOIS database at the DDN NIC, TELNET to the NIC host, NIC.DDN.MIL. There is no need to login. Type "whois" to call up the information retrieval program. Next, type the name of the person, host, domain, network, or mailbox for which you need information. If you are only typing part of the name, end your search string with a period. Type "help" for a more in-depth explanation of what you can search for and how you can search. If you have trouble, send a message to NIC@NIC.DDN.MIL or call 1-800-235-3155. Bug reports can be sent to BUG-WHOIS@NIC.DDN.MIL and suggestions for improvements to the program can be sent to SUGGESTIONS@NIC.DDN.MIL.

How do I become registered in the DDN NIC's WHOIS database?

If you would like to be listed in the WHOIS database, you must have an electronic mailbox accessible from the Internet. First obtain the file NETINFO:USER-TEMPLATE.TXT. You can either retrieve this file via anonymous FTP from NIC.DDN.MIL or get it through electronic mail. To obtain the file via electronic mail, send a message to SERVICE@NIC.DDN.MIL and put the file name in the subject line of the message; that is, "Subject: NETINFO USER-TEMPLATE.TXT". The file will be returned to you overnight.

Fill out the name and address information requested in the file and return it to REGISTRAR@NIC.DDN.MIL. Your application will be processed and you will be added to the database. Unless you are an official Point of Contact for a network entity registered at the DDN NIC, the DDN NIC will not regularly poll you for updates, so you should remember to send corrections to your information as your contact data changes.

How do I use the White Pages at PSI?

Performance Systems International, Inc. (PSI), sponsors a White Pages Pilot Project that collects personnel information from member organizations into a database and provides online access to that data. This effort is based on the OSI X.500 Directory standard.

To access the data, TELNET to `WP.PSI.COM` and login as "fred" (no password is necessary). You may now look up information on participating organizations. The program provides help on usage. For example, typing "help" will show you a list of commands, "manual" will give detailed documentation, and "whois" will provide information regarding how to find references to people. For a list of the organizations that are participating in the pilot project by providing information regarding their members, type "whois -org *".

For more information, send a message to `WP-INFO@PSI.COM`.

How do I use the Knowbot Information Service?

The Knowbot Information Service is a white pages "meta-service" that provides a uniform interface to heterogeneous white pages services in the Internet. Using the Knowbot Information Service, you can form a single query that can search for white pages information from the NIC WHOIS service, the CSNET WHOIS service, the PSI White Pages Pilot Project, and MCI Mail, among others, and have the responses displayed in a single, uniform format.

Currently, the Knowbot Information Service can be accessed through TELNET to port 185 on hosts `nri.reston.va.us` and `sol.bucknell.edu`. From a UNIX host, use "`telnet nri.reston.va.us 185`". There is also an electronic mail interface avaliable by sending mail to netaddress at either `nri.reston.va.us` or `sol.bucknell.edu`.

The commands "help" and "man" summarize the command interface. Simply entering a user name at the prompt searches a default list of Internet directory services for the requested information. Organization and country information can be included thorgh the syntax: "`userid@organization.country`". For example, the queries "`droms@bucknell`" and "`kille@ucl.gb`" are both valid. Note that these are not Domain Names, but rather a syntax to specify an organization and a country for the search.

The default list of directory services currently includes the whois services at the SRI NIC and the CSNET NIC and the white pages service for MCIMail. If an organization is specified, the PSI X.500 service is also searched. Other services can be requested explicitly.

What is Usenet? What is Netnews?

Usenet and Netnews are common names of a distributed computer bulletin board system that some computers on the Internet participate in. It is not strictly an Internet service: many computers not on the Internet also participate. Netnews can be a valuable tool to economize what might otherwise be a large volume of traffic from electronic mailing lists.

How do I get on Usenet? How do I get Netnews on my computer?

To get on Usenet, you must acquire the software, which is available for some computers at no cost from some anonymous FTP sites across the Internet, and you must find an existing Usenet site that is willing to support a connection to your computer. In many cases, this "connection" merely represents additional traffic over existing Internet access channels.

What is anonymous FTP?

Anonymous FTP is a conventional way of allowing you to sign on to a computer on the Internet and copy specified public files from it [RFC-959]. Some sites offer anonymous FTP to distribute software and various kinds of information. You use it like any FTP, but the username is "anonymous". Many systems will allow any password and request that the password you choose is your userid. If this fails, the generic password is usually "guest".

What is "TELNET"?

The term "TELNET" refers to the remote login that's possible on the Internet because of the TELNET Protocol [RFC-854]. The use of this term as a verb, as in "telnet to a host" means to establish a connection across the Internet from one host to another. Usually, you must have an account on the remote host to be able to login to it once you've made a connection. However, some hosts, such as those offering white pages directories, provide public services that do not require a personal account.

RFC-1206 FYI on Questions and Answers for "New Internet Users"

Mailing Lists

What is a mailing list?

A mailing list is really nothing more than an alias that has multiple destinations. Mailing lists are usually created to discuss specific topics. Anybody interested in that topic, may (usually) join that list. Some mailing lists have membership restrictions, others have message content restrictions, and still others are moderated. Most large, "public" mailing lists, such as IETF and TCP-IP, have an additional mail address to which requests to be added or deleted may be sent. Usually, these are of the form listname-request.

There is a "list-of-lists" file available on the host ftp.nisc.sri.com that lists most of the major mailing lists, describes their primary topics, and explains how to subscribe to them. The file is available for anonymous ftp in the netinfo directory as interest-groups (that is, the path is: netinfo/interest-groups). It can also be obtained via electronic mail. Send a message to `mail-server@nisc.sri.com` with the body of the message reading, "Send netinfo/interest-groups" and the file will be returned in moderate size pieces via electronic mail.

How do I contact the administrator of a mailing list rather than posting to the entire list?

For every mailing list mentioned in the "interest-groups" file, there is a description of how to join the list or send other such administrative messages to the person in charge of the list. In general, however, it is usually safe to assume that you can send a message to an address in the format of `ListName-request@domain`. The convention of having a parallel mailbox conforming to the "-request" format is very widely followed. All administrative messages regarding using, joining, or quitting the list should be sent to that mailbox instead of to the whole list so that the readers of the list don't have to read them.

What are some good mailing lists or news groups?

The TCP-IP, IETF, and RFC Distribution lists are primary lists for new Internet users who desire further information about current and emerging developments in the Internet. The first two lists are unmoderated discussion lists, and the latter is an announcement service used by the RFC Editor.

How do I subscribe to the TCP-IP mailing list?

To be added to the TCP-IP mailing list, send a message to:

```
TCP-IP-REQUEST@NIC.DDN.MIL
```

How do I subscribe to the IETF mailing list?

To be added to the IETF mailing list, send a message to:

IETF-REQUEST@ISI.EDU

How do I subscribe to the RFC Distribution list?

To be added to the RFC Distribution list, send a message to:

RFC-REQUEST@NIC.DDN.MIL

RFC-1206 FYI on Questions and Answers for "New Internet Users"

Miscellaneous "Internet lore" Questions

What does :-) mean?

In many electronic mail messages, it is sometimes useful to indicate that part of a message is meant in jest. It is also sometimes useful to communicate emotion which simple words do not readily convey. To provide these nuances, a collection of "smiley faces" has evolved. If you turn your head sideways to the left, :-) appears as a smiling face. Some of the more common faces are:

:-) smile
:) also a smile
:-D laughing
:-} grin
:-] smirk
:-(frown
;-) wink
8-) wide-eyed
:-X close mouthed
:-o oh, no!

What do "btw", "fyi", "imho", "wrt", and "rtfm" mean?

Often commmon expressions are abbreviated in informal network postings. These abbreviations stand for "by the way", "for your information", "in my humble [or honest] opinion", "with respect to", and "read the f*ing manual" (with the "f" word varying according to the vehemence of the reader).

What is the "FAQ" list?

This list provides answers to "Frequently Asked Questions" that often appear on various Usenet newsgroups. The list is posted every four to six weeks to the news.announce.newusers group. It is intended to provide a background for new users learning how to use the news. As the FAQ list provide new users with the answers to such questions, it helps keep the newsgroups themselves comparatively free of repetition. Often specific newsgroups will have and frequently post versions of a FAQ list that are specific to their topics.

Other information is also routinely posted. Here are the subject lines of several general information postings provided on Usenet:

Answers to Frequently Asked Questions (the "FAQ" list)
Introduction to news.announce
Rules for posting to Usenet
How to Create a New Newsgroup
How to Create a New Trial Newsgroup
A Primer on How to Work With the Usenet Community
Emily Postnews Answers Your Questions on Netiquette
Hints on writing style for Usenet
USENET Software: History and Sources
List of Active Newsgroups
Alternative Newsgroup Hierarchies
How to Construct the Mailpaths File

Regional Newsgroup Hierarchies
List of Moderators
Publicly Accessible Mailing Lists
List of Periodic Informational Postings
How to Get Information about Networks
A Guide to Social Newsgroups and Mailing Lists

Suggested Reading

For further information about the Internet and its protocols in general, you may choose to obtain copies of the following works:

Bowers, K., T. LaQuey, J. Reynolds, K. Roubicek, M. Stahl, and A. Yuan, "Where to Start - A Bibliography of General Internetworking Information", RFC 1175, FYI 3, CNRI, U Texas, ISI, BBN, SRI, Mitre, August 1990.

Comer, D., "Internetworking with TCP/IP: Principles, Protocols, and Architecture", Prentice Hall, New Jersey, 1989.

Krol, E., "The Hitchhikers Guide to the Internet", RFC 1118, University of Illinois Urbana, September 1989.

RFC-1206 FYI on Questions and Answers for "New Internet Users"

Condensed Glossary

As with any profession, computers have a particular terminology all their own. Below is a condensed glossary to assist in making some sense of the Internet world.

ACM	Association for Computer Machinery A group established in 1947 to promote professional development and research on computers.
address	There are two separate uses of this term in internet networking: "electronic mail address" and "internet address". An electronic mail address is the string of characters that you must give an electronic mail program to direct a message to a particular person. See "internet address" for its definition.
AI	Artificial Intelligence - The branch of computer science which deals with the simulation of human intelligence by computer systems.
AIX	Advanced Interactive Executive - IBM's version of Unix.
ANSI	American National Standards Institute - A group that certifies organizations which develop U.S. standards for the information processing industry. ANSI accredited groups participate in defining network protocol standards.
ARP	Address Resolution Protocol - An Internet protocol which runs on Ethernet and all IEEE 802.X LANs which maps internet addresses to MAC addresses.
ARPA	Advanced Research Projects Agency - The former name of what is now called DARPA.
ARPANET	Advanced Research Projects Agency Network - A pioneering long haul network funded by ARPA. It served as the basis for early networking research as well as a central backbone during the development of the Internet. The ARPANET consisted of individual packet switching computers interconnected by leased lines.
AS	Autonomous System - A collection of gateways (routers) under a single administrative authority using a common Interior Gateway Protocol for routing packets.
ASCII	American Standard Code for Information Interchange
B	Byte - One character of information, usually eight bits wide.
b	bit - binary digit - The smallest amount of information which may be stored in a computer.
BBN	Bolt Beranek and Newman, Inc. - The Cambridge, MA company responsible for development, operation and monitoring of the ARPANET, and later, the Internet core gateway system, the CSNET Coordination and Information Center (CIC), and NSFNET Network Service Center (NNSC).
BITNET	Because It's Time Network - BITNET has about 2,500 host computers, primarily at universities, in many countries. It is

	managed by EDUCOM, which provides administrative support and information services. There are three main constituents of the network: BITNET in the United States and Mexico, NETNORTH in Canada, and EARN in Europe. There are also AsiaNet, in Japan, and connections in South America. See CREN.
bps	bits per second - A measure of data transmission speed.
BSD	Berkeley Software Distribution - Term used when describing different versions of the Berkeley UNIX software, as in "4.3BSD UNIX".
catenet	A network in which hosts are connected to networks with varying characteristics, and the networks are interconnected by gateways (routers). The Internet is an example of a catenet.
CCITT	International Telegraph and Telephone Consultative Committee
core g'way	Historically, one of a set of gateways (routers) operated by the Internet Network Operations Center at BBN. The core gateway system forms a central part of Internet routing in that all groups had to advertise paths to their networks from a core gateway.
CREN	The Corporation for Research and Educational Networking - BITNET and CSNET have recently merged to form CREN.
CSNET	Computer + Science Network - A large data communications network for institutions doing research in computer science. It uses several different protocols including some of its own. CSNET sites include universities, research laboratories, and commercial companies. See CREN.
DARPA	U.S. Department of Defense Advanced Research Projects Agency - The government agency that funded the ARPANET and later started the Internet.
datagram	The unit transmitted between a pair of internet modules. The Internet Protocol provides for transmitting blocks of data, called datagrams, from sources to destinations. The Internet Protocol does not provide a reliable communication facility. There are no acknowledgements either end-to-end or hop-by-hop. There is no error control for data, only a header checksum. There are no retransmissions. There is no flow control. See IP.
DCA	Defense Communications Agency - The government agency responsible for installation of the Defense Data Network (DDN), including the ARPANET and MILNET lines and PSNs. Currently, DCA administers the DDN, and supports the user assistance and network registration services of the DDN NIC.
DDN	Defense Data Network - Comprises the MILNET and several other DoD networks.
DDN NIC	The network information center at SRI International. It is the primary repository for RFCs and Internet Drafts, as well as providing other services.
DEC	Digital Equipment Corporation

DECnet	Digital Equipment Corporation network - A networking protocol for DEC computers and network devices.
default route	A routing table entry which is used to direct any data addressed to any network numbers not explicitly listed in the routing table.
DNS	The Domain Name System is a mechanism used in the Internet for translating names of host computers into addresses. The DNS also allows host computers not directly on the Internet to have registered names in the same style, but returns the electronic mail gateway which accesses the non-Internet network instead of an IP address.
DOD	U.S. Department of Defense
DOE	U.S. Department of Energy
dot addr.	(dotted address notation) - Dot address refers to the common notation for Internet addresses of the form A.B.C.D; where each letter represents, in decimal, one byte of the four byte IP address.
EARN	European Academic Research Network - One of three main constituents of BITNET.
EBCDIC	Extended Binary-coded Decimal Interchange Code
EGP	Exterior Gateway Protocol - A protocol which distributes routing information to the gateways (routers) which connect autonomous systems.
Ethernet	A network standard for the hardware and data link levels. There are two types of Ethernet: Digital/Intel/Xerox (DIX) and IEEE 802.3.
FDDI	Fiber Distributed Data Interface - FDDI is a high-speed (100Mb) token ring LAN.
FIPS	Federal Information Processing Standard
FTP	File Transfer Protocol - The Internet standard high-level protocol for transferring files from one computer to another.
gateway	See router
GB	Gigabyte - A unit of data storage size which represents 2^30 (over 1 billion) characters of information.
Gb	Gigabit - 2^30 bits of information (usually used to express a data transfer rate; as in, 1 gigabit/second = 1Gbps).
GNU	Gnu's Not UNIX - A UNIX-compatible operating system developed by the Free Software Foundation.
header	The portion of a packet, preceding the actual data, containing source and destination addresses and error-checking fields.
host number	The part of an internet address that designates which node on the (sub)network is being addressed.
HP	Hewlett-Packard
HYPERchannel	High-speed communications link.
I/O	Input/Output

IAB — Internet Activities Board - The IAB is the coordinating committee for Internet design, engineering and management.

IBM — International Business Machines Corporation

ICMP — Internet Control Message Protocol - ICMP is an extension to the Internet Protocol. It allows for the generation of error messages, test packets and informational messages related to IP.

IEEE — Institute for Electrical and Electronics Engineers

IETF — Internet Engineering Task Force - The IETF is a large open community of network designers, operators, vendors, and researchers whose purpose is to coordinate the operation, management and evolution of the Internet, and to resolve short- and mid-range protocol and architectural issues. It is a major source of proposed protocol standards which are submitted to the Internet Activities Board for final approval. The IETF meets three times a year and extensive minutes of the plenary proceedings are issued.

internet(work) — Any connection of two or more local or wide-area networks.

Internet — The global collection of interconnected local, mid-level and wide-area networks which use IP as the network layer protocol.

internet address — An assigned number which identifies a host in an internet. It has two or three parts: network number, optional subnet number, and host number.

IP — Internet Protocol - The network layer protocol for the Internet. It is a packet switching, datagram protocol defined in RFC 791.

IRTF — Internet Research Task Force - The IRTF is a community of network researchers, generally with an Internet focus. The work of the IRTF is governed by its Internet Research Steering Group (IRSG).

ISO — International Organization for Standardization

KB — Kilobyte - A unit of data storage size which represents 2^10 (1024) characters of information.

Kb — Kilobit - 2^10 bits of information (usually used to express a data transfer rate; as in, 1 kilobit/second = 1Kbps = 1Kb).

LAN — Local Area Network - A network that takes advantage of the proximity of computers to offer relatively efficient, higher speed communications than long-haul or wide-area networks.

LISP — List Processing Language - A high-level computer language invented by Professor John McCarthy in 1961 to support research into computer based logic, logical reasoning, and artificial intelligence. It was the first symbolic (as opposed to numeric) computer processing language.

MAC — Medium Access Control - For broadcast networks, it is the method which devices use to determine which device has line access at any given time.

Mac — Apple Macintosh computer.

MAN — Metropolitan Area Network

MB	Megabyte - A unit of data storage size which represents 2^20 (over one million) characters of information.
Mb	Megabit - 2^20 bits of information (usually used to express a data transfer rate; as in, 1 megabit/second = 1Mbps).
MILNET	Military Network - A network used for unclassified military production applications. It is part of the DDN and the Internet.
MIT	Massachusetts Institute of Technology
MTTF	Mean Time to Failure - The average time between hardware breakdown or loss of service. This may be an empirical measurement or a calculation based on the MTTF of component parts.
MTTR	Mean Time to Recovery (or Repair) - The average time it takes to restore service after a breakdown or loss. This is usually an empirical measurement.
MVS	Multiple Virtual Storage - An IBM operating system based on OS/1.
NASA	National Aeronautics and Space Administration
NBS	National Bureau of Standards - Now called NIST.
network number	The part of an internet address which designates the network to which the addressed node belongs.
NFS	Network File System - A network service that lets a program running on one computer to use data stored on a different computer on the same internet as if it were on its own disk.
NIC	Network Information Center - An organization which provides network users with information about services provided by the network.
NOC	Network Operations Center - An organization that is responsible for maintaining a network.
NIST	National Institute of Standards and Technology - Formerly NBS.
NSF	National Science Foundation
NSFNET	National Science Foundation Network - The NSFNET is a highspeed "network of networks" which is hierarchical in nature. At the highest level is a network that spans the continental United States. Attached to that are mid-level networks and attached to the mid-levels are campus and local networks. NSFNET also has connections out of the U.S. to Canada, Mexico, Europe, and the Pacific Rim. The NSFNET is part of the Internet.
NSFNET	Mid-level Level Network - A network connected to the highest level of the NSFNET that covers a region of the United States. It is to mid-level networks that local sites connect. The mid-level networks were once called "regionals".
OSI	Open Systems Interconnection - A set of protocols designed to be an international standard method for connecting unlike computers and networks. Europe has done most of the work developing OSI and will probably use it as soon as possible.
OSI Ref. Model	An "outline" of OSI which defines its seven layers and their

	functions. Sometimes used to help describe other networks.
OSPF	Open Shortest-Path First Interior Gateway Protocol - A proposed replacement for RIP. It addresses some problems of RIP and is based upon principles that have been well-tested in non-internet protocols. Originally acronymed as OSPFIGP.
packet	The unit of data sent across a packet switching network. The term is used loosely. While some Internet literature uses it to refer specifically to data sent across a physical network, other literature views the Internet as a packet switching network and describes IP datagrams as packets.
PC	Personal Computer
PCNFS	Personal Computer Network File System
POSIX	Portable Operating System Interface - Operating system based on UNIX.
PPP	Point-to-Point Protocol - The Point-to-Point Protocol (PPP) provides a method for transmitting datagrams over serial point-to-point links.
protocol	A formal description of message formats and the rules two computers must follow to exchange those messages. Protocols can describe low-level details of machine-to-machine interfaces (e.g., the order in which bits and bytes are sent across a wire) or high-level exchanges between allocation programs (e.g., the way in which two programs transfer a file across the Internet).
RFC	The Internet's Request for Comments documents series. The RFCs are working notes of the Internet research and development community. A document in this series may be on essentially any topic related to computer communication, and may be anything from a meeting report to the specification of a standard.
RIP	Routing Interchange Protocol - One protocol which may be used on internets simply to pass routing information between gateways. It is used on may LANs and on some of the NSFNET intermediate level networks.
RJE	Remote Job Entry - The general protocol for submitting batch jobs and retrieving the results.
RLOGIN	Remote Login - A service on internets very similar to TELNET. RLOGIN was invented for use between Berkeley Unix systems on the same LAN at a time when TELNET programs didn't provide all the services users wanted. Berkeley plans to phase it out.
router	A special-purpose dedicated computer that attaches to two or more networks and routes packets from one network to the other. In particular, an Internet gateway routes IP datagrams among the networks it connects. Gateways route packets to other gateways until they can be delivered to the final destination directly across one physical network.
RPC	Remote Procedure Call - An easy and popular paradigm for implementing the client-server model of distributed computing.

server	A computer that shares its resources, such as printers and files, with other computers on the network. An example of this is a Network Files System (NFS) Server which shares its disk space with one or more workstations that may not have local disk drives of their own.
SLIP	Serial Line Internet Protocol - SLIP is currently a defacto standard, commonly used for point-to-point serial connections running TCP/IP. It is not an Internet standard but is defined in RFC 1055.
SMTP	Simple Mail Transfer Protocol - The Internet standard protocol for transferring electronic mail messages from one computer to another. SMTP specifies how two mail systems interact and the format of control messages they exchange to transfer mail.
SNA	System Network Architecture - IBM's data communications protocol.
SNMP	Simple Network Management Protocol - The Simple Network Management Protocol (RFC 1157) is the Internet's standard for remote monitoring and management of hosts, routers and other nodes and devices on a network.
subnet	A portion of a network, which may be a physically independent network, which shares a network address with other portions of the network and is distinguished by a subnet number. A subnet is to a network what a network is to an internet.
subnet number	A part of the internet address which designates a subnet. It is ignored for the purposes internet routing, but is used for intranet routing.
T1	A term for a digital carrier facility used to transmit a DS-1 formatted digital signal at 1.544 megabits per second.
T3	A term for a digital carrier facility used to transmit a DS-3 formatted digital signal at 44.746 megabits per second.
TCP	Transmission Control Protocol - A transport layer protocol for the Internet. It is a connection oriented, stream protocol defined by RFC 793.
TCP/IP	Transmission Control Protocol/Internet Protocol This is a common shorthand which refers to the suite of application and transport protocols which run over IP. These include FTP, TELNET, SMTP, and UDP (a transport layer protocol).
Telenet	A public packet-switching network operated by US Sprint. Also known as "SprintNet".
TELNET	The Internet standard protocol for remote terminal connection service. TELNET allows a user at one site to interact with a remote timesharing system at another site as if the user's terminal was connected directly to the remote computer.
THEnet	The Texas Higher Education Network, a multiprotocol network connecting most major academic and research institutions in the State of Texas, as well as several institutions in Mexico.
Token Ring	A type of LAN. Examples are IEEE 802.5, ProNET-10/80 and FDDI. The term "token ring" is often used to denote 802.5

Tymnet	A public character-switching/packet-switching network operated by British Telecom.
UDP	User Datagram Protocol - A transport layer protocol for the Internet. It is a datagram protocol which adds a level of reliability and multiplexing to IP datagrams. It is defined in RFC 768.
ULTRIX	UNIX-based operating system for Digital Equipment Corporation computers.
UNIX	An operating system developed by Bell Laboratories that supports multiuser and multitasking operations.
UUCP	UNIX-to-UNIX Copy Program - A protocol used for communication between consenting UNIX systems.
VMS	Virtual Memory System - A Digital Equipment Corporation operating system.
WAN	Wide Area Network
WHOIS	An Internet program which allows users to query a database of people and other Internet entities, such as domains, networks, and hosts, kept at the NIC. The information for people shows a person's company name, address, phone number and email address.
XNS	Xerox Network System - A data communications protocol suite developed by Xerox. It uses Ethernet to move the data between computers.
X.25	A data communications interface specification developed to describe how data passes into and out of public data communications networks. The public networks such as Sprintnet and Tymnet use X.25 to interface to customer computers.

Authors' Addresses

Gary Scott Malkin
FTP Software, Inc.
26 Princess Street
Wakefield, MA 01880

Phone: (617) 246-0900

EMail: gmalkin@ftp.com

April N. Marine
SRI International
Network Information Systems Center
333 Ravenswood Avenue, EJ294
Menlo Park, CA 94025

Phone: (415) 859-5318

EMail: APRIL@nic.ddn.mil

RFC-1207: FYI on Questions and Answers to Commonly Asked "Experienced Internet User" Questions

G. Malkin, FTP Software, Inc.
A. Marine, SRI
J. Reynolds, ISI
February 1991
FYI: 7

This FYI RFC is one of two FYI's called, "Questions and Answers" (Q/A), produced by the User Services Working Group of the Internet Engineering Task Force (IETF). The goal is to document the most commonly asked questions and answers in the Internet.

This memo provides information for the Internet community. It does not specify any standard. Distribution of this memo is unlimited. Security issues are not discussed in this memo.

RFC-1207 FYI on Questions and Answers for "Experienced Internet Users"

Introduction

During the last few months, several people have monitored various major mailing lists and have extracted questions that are important or commonly asked. This FYI RFC is one of two in a series of FYI's which present the questions and their answers. The first FYI, FYI 4, presented questions new Internet users commonly ask and their answers.

The goal of this FYI is to codify the Internet lore so that network operations staff, especially for networks just joining the Internet, will have an accurate and up to date set of references from which to work. Also, redundancies are moved away from the electronic mailing lists so that the lists' subscribers do not have to read the same queries and answers over and over again.

Although the questions and their responses are taken from various mailing lists, they are presented here loosely grouped by related topic for ease of reading. First the question is presented, then the answer (or answers) as it appeared on the mailing list.

Sometimes the answers are abridged for better use of space. If a question was not answered on the mailing list, the editors provide an answer. These answers are not distinguished from the answers found on the lists. Sometimes, in order to be as complete as possible, the editors provide additional information that was not present in the original answer. If so, that information falls under the heading "Additional Information".

The answers are as correct as the reviewers can make them. However, much of this information changes with time. As the FYI is updated, temporal errors will be corrected.

Many of the questions are in first person, and the answers were directed to the originator of the question. These phrasings have not been changed except where necessary for clarity. References to the correspondents' names have been removed.

The Q/A mailing lists are maintained by Gary Malkin at FTP.COM. They are used by a subgroup of the User Services Working Group to discuss the Q/A FYIs. They include:

`quail@ftp.com` This is a discussion mailing list. Its primary use is for pre-release review of the Q/A FYIs.

`quail-request@ftp.co` This is how you join the quail mailing list.

`quail-box@ftp.com` This is where the questions and answers will be forwarded-and-stored. It is not necessary to be on the quail mailing list to forward to the quail-box.

Acknowledgments

The following people deserve thanks for their help and contributions to this FYI Q/A: Jim Conklin (EDUCOM), John C. Klensin (MIT), Professor Kynikos (Special Consultant), Jon Postel (ISI), Marshall Rose (PSI, Inc.), David Sitman (Tel Aviv University), Patricia Smith (Merit), Gene Spafford (Purdue), and James Van Bokkelen (FTP Software, Inc.).

Questions about the Internet

How do I get statistics regarding the traffic on NSFNET?

Merit/NSFNET Information Services maintains a variety of statistical data at 'nis.nsf.net' (35.1.1.48) in the 'stats' directory. Information includes packet counts by NSS and byte counts for type of use (ftp, smtp, telnet, etc.). Filenames are of the form 'NSFyy-mm.type'.

Files are available for anonymous ftp; use 'guest' as the password.

The data in these files represent only traffic which traverses the highest level of the NSFNET, not traffic within a campus or regional network. Send questions/comments to nsfnet- info@merit.edu.

RFC-1207 FYI on Questions and Answers for "Experienced Internet Users"

Questions About Other Networks and Internets

We have a user who would like to access a machine on "EARN/BITNET". I can't find anything on this in the domain name tables. Please, what is this, and how do I connect to it?

> There are several machines on the Internet that act as gateways between the Internet and BITNET. Two examples are UICVM.UIC.EDU and CUNYVM.CUNY.EDU. You can address a mail message to user %nodename.bitnet@uicvm.uic.edu where the message will be passed from the Internet to BITNET.

Additional Information:

These same gateways, known as INTERBIT on the BITNET/EARN side, transfer mail from computers on that network which support SMTP mail headers, onto the Internet. (Many BITNET/EARN computers still do not support SMTP, which is not a part of the IBM protocol used, and it is not possible to send mail from those computers across the gateways into the Internet, in general.)

BITNET and EARN are the two largest of several cooperating networks which use the IBM RSCS/NJE protocol suite, but are not limited to IBM systems. These independently administered, interconnected networks function as a single, worldwide network directly connecting more than 3,300 computers in about 1,400, mostly higher-education, organizations worldwide. This worldwide network supports electronic mail, including mailing lists, sender-initiated file transfer, and short "interactive" messages.

BITNET, frequently used (outside of Europe) to refer to the whole worldwide network, technically refers to that portion in the United States, plus sites in other countries which are connected through the United States and do not have their own separately administered cooperating networks. More than 550 organizations in the U.S. participate in BITNET.

EARN is the European Academic Research Network. EARN links more than 500 institutions in Europe and several surrounding countries.

BITNET and CSNET merged organizationally on October 1, 1990, to form CREN, the Corporation for Research and Educational Networking. The two networks remain separate at the operational level level, however. (EARN and the other Cooperating Networks were not involved in this merger.)

RFC-1207 FYI on Questions and Answers for "Experienced Internet Users"

Questions About Internet Documentation

Where do I get information regarding ordering documents related to GOSIP?

The complete information as issued by NIST is available online on the NIC.DDN.MIL host as PROTOCOLS:GOSIP-ORDER-INFO.TXT. The file contains pointers to contact people, ordering addresses, prices, and, in some cases, online pathnames, for various GOSIP related documents. In addition, the information as of August 1990 was published as an appendix to RFC-1169, "Explaining the Role of GOSIP".

RFC-1207 FYI on Questions and Answers for "Experienced Internet Users"

Questions About Domain Name System (DNS)

RFC-1207 FYI on Questions and Answers for "Experienced Internet Users"

Questions About Domain Name System (DNS)

Is there a DNS Query server?

Actually, what you are looking for is the service that host 128.218.1.109 provides on port 5555 - you simply connect to that host at that port, type in a fully qualified domain name and it responds with an internet address and closes the connection. I used it when I had a host that still only had /etc/hosts and it did just what I needed - which was basically a manual nslookup.

However, the vast majority of users will find it simpler to just use a DNS query tool and ask the DNS directly. This doesn't require much sophistication, and does allow the user to see how short names are expanded at the user's site rather than at 128.218.1.109 (wherever that is). For example, suppose a user wants to find out the address of a fully-qualified domain name "X.MISKATONIC.EDU", and also see what host and address are used when "Z" is typed as a host name.

Assuming the user is on a UNIX host and has a copy of the dig program, type:

dig x.miskatonic.edu

and

dig z

and the answers will appear. You are now on your way to becoming a DNS expert. There are other UNIX alternatives, e.g., nslookup, and similar programs for non-UNIX systems. Your local DNS guru certainly has one or more of these tools, and although they are often kept from the public, they are really quite easy to use for simple cases.

RFC-1207 FYI on Questions and Answers for "Experienced Internet Users"

Questions About Domain Name System (DNS)

Has any one else had frequent BIND failures (especially major domain sites that have heavy TCP domain loads)?

We have been having a frequent BIND failure on both our VAX and Solbourne that is traced to TCP domain queries from an IBM NSMAIN nameserver running in cache mode (UDP queries do not cause this problem, though it is usually a UDP resolution that is active upon the crash -- this resolution is an innocent victim).

I have discovered that something is trashing the hash areas (sometimes even as it is being recursively used in a resolution). Also, occasionally the socket/file descriptor for the TCP connection is changed to invalid entries causing a reply write fail (though this is not necessarily fatal, and the rest of the structure is not apparently altered).

In both the case of BIND and the IBM implementation, often called FAL, there are multiple versions, with older versions being truly bad. Upgrade to recent version before exploring further.

BIND has always had a problem with polluting its own database.

These problems have been related to TCP connections, NS RRs with small TTLs, and several other causes. Experience suggests that the style of bug fixing has often been that of reducing the problem by 90% rather than eliminating it.

IBM's support for the DNS (outside of UNIX systems) is interesting in its techniques, encouraging in its improvement, but still somewhat depressing when compared to most other DNS software. IBM also uses terminology that varies somewhat from the usual DNS usage and preserves some archaic syntax, e.g., "..".

The combination of an old BIND and an old IBM server is just plain unpleasant.

Questions About Domain Name System (DNS)

Is the model used by the domain name system for host names that the owner of a name gets to choose its case?

> The model used by the DNS is that you get to control at a specific point in the name space, and are hence free to select case as you choose, until points where you in turn give away control. As a practical matter, there are several implementations that don't do the right thing. IBM implementations often map everything into a single case.

RFC-1207 FYI on Questions and Answers for "Experienced Internet Users"

Questions About Domain Name System (DNS)

According to RFC 1034 (Name Server Technical Considerations) one should not have to code glue RR's for name server's names unless they are below the cut. When I don't put glue RR's in, and do a query for NS records, the "additional" field is left blank. As far as I can tell, all other zones I query for NS records have this filled with the IP addresses of the NS hosts. Is this required or should I not be concerned that the additional field is empty?

> The protocol says that an empty additional field is not a problem when the name server's name is not "below" the cut.
>
> In practice, putting in the glue where it is not required can cause problems if the servers named in the glue are used for several zones. This is broken behavior in BIND. Not putting in glue can cause other problems in BIND, usually when the server name is difficult to resolve. So, the bottom line is to put glue in only when required, and don't use aliases or anything else tricky when it comes to identifying name servers.

RFC-1207 FYI on Questions and Answers for "Experienced Internet Users"

Questions About Network Management Implementations

Questions About Network Management Implementations

In reading the SNMP RFCs (1155-1158) I find mention of authentication of PDUs. Are there any standards for authentication mechanisms?

> There is a working group of the IETF that is working on this problem. They are close to a solution, but nothing has yet reached RFC publication yet. Expect something solid and implementable by October of 1991.

Questions About Network Management Implementations

Can vendors make their enterprise-specific variables available to users through a standard distribution mechanism?

Yes. But before someone submits a MIB, they should check it out themselves.

On uu.psi.com in pilot/snmp-wg/, there are two files

```
mosy-sparc-4.0.3.c
```

```
mosy-sun3-3.5
```

The first will run on a Sun-Sparc, the second will run on a Sun-3. After retrieving one of these files in BINARY mode via anonymous FTP, the submittor can run their MIB through it, e.g.,

```
% mosy mymib.my
```

Once your MIB passes, send it to:

```
mib-checker@isi.edu
```

If everything is OK, the mib-checker will arrange to have it installed in the /share/ftp/mib directory on venera.isi.edu.

Note: This processing does not offer an official endorsement. The documents submitted must not be marked proprietary, confidential, or the like.

RFC-1207 FYI on Questions and Answers for "Experienced Internet Users"

Questions About Network Management Implementations

I have a question regarding those pesky octet strings again. I use the variable-type field of the Response pdu to determine how the result should be displayed to the user. For example, I convert NetworkAddresses to their dotted decimal format ("132.243.50.4"). I convert Object Identifiers into strings ("1.3.6.1.2....").

I would LIKE to just print Octet Strings as strings. But, this causes a problem in such cases as atPhysAddress in which the Octet string contains the 6 byte address instead of a printable ASCII string. In this case, I would want to display the 6 bytes instead of just trying to print the string.

MY QUESTION IS: Does anyone have a suggestion as to how I can determine whether I can just print the string or whether I should display the octet bytes. * Remember: I want to support enterprise specific variables too.

> In general, there is no way that you can tell what is inside an OCTET STRING without knowing something about the object that the OCTET STRING comes from. In MIB-II, (RFC-1158), some objects are marked as DisplayString which has the syntax of OCTET STRING but is restricted to characters from the NVT ASCII character set (see the TELNET Specification, RFC 854, for further information). These objects are:
>
> ```
> sysDescr
> sysContact
> sysName
> sysLocation
> ifDescr
> ```
>
> If you want to be able to arbitrarily decide how to display the strings, without knowing anything about the object, then you can scan the octets, looking for any octet which is not printable ASCII. If you find at least one, you can print the entire string, octet by octet, in "%02x:" notation. If all of the octets are printable ASCII, then you can just printf the string.

Questions About Network Management Implementations

If archived MIBs must be 1155-compatible ("Structure and Identification of Management Information for TCP/IP-based Internets")**, it would be nice if those who submit them check them first. Where are these MIB tools available for public FTP? Ideally, a simple syntax checker (that didn't actually generate code) would be nice.**

> In the ISODE 6.0 release there is a tool called MOSY which recognizes the 1155 syntax and produces a flat ASCII file. If you can run it through MOSY without problems then you are OK.

RFC-1207 FYI on Questions and Answers for "Experienced Internet Users"

Questions About Network Management Implementations

Suppose I want to create a private MIB object for causing some action to happen, say, do a reset. Should the syntax or this object specify a value such as:

Syntax:

```
INTEGER {
perform reset (1),
}
```

even though there is only a single value? Or, is it ok to just allow a Set on this object with any value to perform the desired action? If the later, how is this specified?

For our SNMP manageable gizmos and doohickies with similar "action" type MIB variables, I've defined two values

Syntax:

```
INTEGER {
reset(1)
not-reset(2)
}
```

And defined behavior so that the only valid value that the variable may be set to is "reset" (which is returned in the get response PDU) and at all other times a get/getnext will respond with "not-reset".

RFC-1207 FYI on Questions and Answers for "Experienced Internet Users"

Questions about Serial Line Internet Protocol (SLIP) and Point-to-Point Protocol (PPP) Implementations

RFC-1207 FYI on Questions and Answers for "Experienced Internet Users"

Questions about Serial Line Internet Protocol (SLIP) and Point-to-Point Protocol (PPP) Implementations

I seem to recall hearing that SLIP will only run on synchronous serial lines. Is this true? ... is there something about SLIP which precludes it's being implemented over async lines?

> Other way around: SLIP is designed for async lines and is not a good fit on sync lines. PPP works on either, and is what you should be implementing if you're implementing something.

RFC-1207 FYI on Questions and Answers for "Experienced Internet Users"

Questions about Serial Line Internet Protocol (SLIP) and Point-to-Point Protocol (PPP) Implementations

Since we are very interested in standards in this area, could someone tell me where I can find more information on PPP?

Also, can this protocol be used in other fields than for the Internet (i.e., telecontrol, telemetering) where we see a profusion of proprietary incompatible and hard to maintain Point-to-Point Protocols?

PPP was designed to be useful for many protocols besides just IP. Whether it would be useful for your particular application should probably be discussed with the IETF's Point-to-Point Protocol Working Group discussion list. For general discussion: `ietf- ppp@ucdavis.edu`. To subscribe: `ietf-ppp-request@ucdavis.edu`

The PPP specification is available as RFC-1171, and a PPP options specification is available as RFC-1172.

In UnixWorld of April 1990 (Vol. VII, No. 4, Pg. 85), Howard Baldwin writes:

"Point-to-Point Protocol (PPP) has just been submitted to the CCITT from the Internet Engineering Task Force. It specifies a standard for encapsulating Internet Protocol data and other network layer (level three on ISO's OSI Model) protocol information over point-to-point links; it also provides ways to test and configure lines and the upper level protocols on the OSI Model. The only requirement is a provision of a duplex circuit either dedicated or switched, that can operate in either an asynchronous or synchronous mode, transparent to the data-linklayer frame.

"According to Michael Ballard, director of network systems for Telebit, PPP is a direct improvement upon Serial Line Internet Protocol (SLIP), which had neither error correction nor a way to exchange network address."

Questions about Serial Line Internet Protocol (SLIP) and Point-to-Point Protocol (PPP) Implementations

Does anyone know if there is a way to run a SLIP program on a IBM computer running SCO Xenix/Unix, with a multi-port serial board?

SCO TCP/IP for Xenix supports SLIP. It works. However, be warned: SCO SLIP works *only* with SCO serial drivers, so it will *not* work with intelligent boards that come with their own drivers. If you want lots of SLIP ports, you'll need lots of dumb ports, perhaps with a multi-dumb-port board.

Here's the setup -- SunOS 3.5, with the 4.3BSD TCP, IP & SLIP distributions installed. Slip is running between the "ttya" ports of two Sun 3/60's. "ping", "rlogin", etc., works fine, but a NFS mount results in "server not responding: RPC Timed Out".

SunOS 3.5 turns the UDP checksum off, which is legal and works okay over interfaces such as ethernet which has link- level checksumming. On the other hand, SLIP doesn't perform checksums thus running NFS over SLIP requires you to turn the UDP checksum on. Otherwise, you'll experience erratic behavior such as the one described above.

Save the older kernel and try,

```
% adb -k -w /vmunix /dev/kmem udpcksum?w 1
```

to patch up the kernel.

Questions About Routing

Some postings mentioned "maximum entropy routing". Could someone please provide a pointer to on-line or off-line references to this topic?

> Try NYU CSD Technical Report 371: "Some Comments on Highly Dynamic Network Routing," by Herbert J. Bernstein, May 1988.

Other Protocol and Standards Implementation Questions

Other Protocol and Standards Implementation Questions

Does anyone recognize ethernet type "80F3"? I don't see it in RFC 1010, but I am seeing it on our net.

Ethernet type 0x80F3 is used by AppleTalk for address resolution. You must have Macs on your network which are directly connected to Ethernet. These packets are used by the Mac (generally at startup) to determine a valid AppleTalk node number.

Additional Information:

RFC 1010 is obsolete. Please consult RFC-1060, the current "Assigned Numbers" (issued March 1990), which does list "80F3":

Ethernet		Exp. Ethernet		Description	References
decimal	Hex	decimal	octal		
33011	80F3	-	-	AppleTalk AARP (Kinetics)	[XEROX]

Other Protocol and Standards Implementation Questions

Does anyone know the significance of a high value for "Bad proto" in the output from netstat on Unix machines using ethernet? We're seeing values in the tens of thousands out of a few hundred thousand packets sent/received in all. Some "Bad proto" values are negative, too. (Off the scale?) Any help would be appreciated.

> This probably indicates that you are getting tens of thousands of broadcast packets from some host or hosts on your network. You might want to buy or rent a LAN monitor, or install one of the public-domain packages to see what private protocol is guilty. "FYI on a Network Management Tool Catalog: Tools for Monitoring and Debugging TCP/IP Internets and Interconnected Devices" (RFC 1147, FYI 2), contains pointers to tools that may help you zero in on the problem.

Other Protocol and Standards Implementation Questions

Which RFC would explain the proper way to configure broadcast addresses when using subnets?

Consult RFC-922, "Broadcasting Internet Datagrams in the Presence of Subnets".

Other Protocol and Standards Implementation Questions

Can anyone tell me what .TAR files exactly are? Is it like ZIP or LZH for the IBM PC's? IF so, how do I go about getting a compressor/decompressor for .TAR files and what computer does this run on?

TAR stands for "Tape ARchive". It is a Unix utility which takes files, and directories of files, and creates a single large file. Originally intended to back up directory trees onto tape (hence the name), TAR is also used to combine files for easier electronic file transfer.

Suggested Reading

For further information about the Internet and its protocols in general, you may choose to obtain copies of the following works:

Bowers, K., T. LaQuey, J. Reynolds, K. Roubicek, M. Stahl, and A. Yuan, "Where to Start - A Bibliography of General Internetworking Information", RFC 1175, FYI 3, CNRI, U Texas, ISI, BBN, SRI, Mitre, August 1990.

Braden, R., Editor, "Requirements for Internet Hosts -- Communication Layer", RFC 1122, Internet Engineering Task Force, October 1989.

Braden, R., Editor, "Requirements for Internet Hosts -- Application and Support", RFC 1123, Internet Engineering Task Force, October 1989.

Comer, D., "Internetworking with TCP/IP: Principles, Protocols, and Architecture", Prentice Hall, New Jersey, 1989.

Frey, D. and R. Adams, "!%@:: A Directory of Electronic Mail Addressing and Networks", O'Reilly and Associates, Newton, MA, August 1989.

Krol, E., "The Hitchhikers Guide to the Internet", RFC 1118, University of Illinois Urbana, September 1989.

LaQuey, T, Editor, "Users' Directory of Computer Networks", Digital Press, Bedford, MA, 1990.

Malkin, G., and A. Marine, "FYI on Questions and Answers - Answers to Commonly asked "New Internet User" Questions", RFC 1206, FYI 4, FTP Software, Inc., SRI, February 1991.

Postel, J., Editor, "IAB Official Protocol Standards", RFC 1140, Internet Activities Board, May 1990.

Quarterman, J., "Matrix: Computer Networks and Conferencing Systems Worldwide", Digital Press, Bedford, MA, 1989.

Reynolds, J., and J. Postel, "Assigned Numbers", RFC 1060, USC/Information Sciences Institute, March 1990.

Socolofsky, T., and C. Kale, "A TCP/IP Tutorial", RFC 1180, Spider Systems Limited, January 1991.

Stevens, W., "UNIX Network Programming", ISBN 0-13-949876-1, Prentice Hall, Englewood Cliffs, NJ, 1990.

Stine, R., Editor, "FYI on a Network Management Tool Catalog: Tools for Monitoring and Debugging TCP/IP Internets and Interconnected Devices" RFC 1147, FYI 2, Sparta, Inc., April 1990.

References

[1] Cerf, V., and K. Mills, "Explaining the Role of GOSIP", RFC 1169, IAB, NIST, August 1990.

[2] Mockapetris, P., "Domain Names - Concepts and Facilities", RFC 1034, USC/Information Sciences Institute, November 1987.

[3] Rose, M., and K. McCloghrie, "Structure and Identification of Management Information for TCP/IP-based Internets", RFC 1155, Performance Systems International, Hughes LAN Systems, May 1990.

[4] McCloghrie, K., and M. Rose, "Management Information Base for Network Management of TCP/IP-based internets", RFC 1156, Hughes LAN Systems, Performance Systems International, May 1990.

[5] Case, J., M. Fedor, M. Schoffstall, and J. Davin, "A Simple Network Management Protocol (SNMP)", RFC 1157, SNMP Research, Performance Systems International, Performance Systems International, MIT Laboratory for Computer Science, May 1990.

[6] Rose, M., Editor, "Management Information Base for Network Management of TCP/IP-based internets: MIB-II", RFC 1158, Performance Systems International, May 1990.

[7] Postel, J., and J. Reynolds, "TELNET Protocol Specification", RFC 854, USC/Information Sciences Institute, May 1983.

[8] Romkey, J., "A Nonstandard for Transmission of IP Datagrams over Serial Lines: SLIP", RFC 1055, June 1988.

[9] Perkins, D., "The Point-to-Point Protocol: A Proposal for Multi- Protocol Transmission of Datagrams Over Point-to-Point Links", RFC 1171, CMU, July 1990.

[10] Perkins, D., and R. Hobby, "The Point-to-Point Protocol (PPP) Initial Configuration Options", CMU, UC Davis, July 1990.

[11] Reynolds, J., and J. Postel, "Assigned Numbers", RFC 1060, USC/Information Sciences Institute, March 1990.

[12] Stine, R., Editor, "FYI on a Network Management Tool Catalog: Tools for Monitoring and Debugging TCP/IP Internets and Interconnected Devices" RFC 1147, FYI 2, Sparta, Inc., April 1990.

[13] Braden, R., Editor, "Requirements for Internet Hosts -- Communication Layer", RFC 1122, Internet Engineering Task Force, October 1989.

Authors' Addresses

Gary Scott Malkin
FTP Software, Inc.
26 Princess Street
Wakefield, MA 01880

Phone: (617) 246-0900
EMail: gmalkin@ftp.com

April N. Marine
SRI International
Network Information Systems Center
333 Ravenswood Avenue, EJ294
Menlo Park, CA 94025

Phone: (415) 859-5318
EMail: APRIL@nic.ddn.mil

Joyce K. Reynolds
USC/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292-6695

Phone: (213) 822-1511
EMail: jkrey@isi.edu

Perkins, D., "The Point-to-Point Protocol: A Proposal for Multi- Protocol Transmission of Datagrams Over Point-to-Point Links", RFC 1171, CMU, July 1990.

Perkins, D., and R. Hobby, "The Point-to-Point Protocol (PPP) Initial Configuration Options", CMU, UC Davis, July 1990.

RFC-1212 Concise MIB Definitions

Marshall Rose & Keith McCloghrie, Editors

March 1991

Status of this Memo

This memo defines a format for producing MIB modules. This RFC specifies an IAB standards track document for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "IAB Official Protocol Standards" for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Abstract

This memo describes a straight-forward approach toward producing concise, yet descriptive, MIB modules. It is intended that all future MIB modules be written in this format.

Historical Perspective

As reported in RFC-1052, IAB Recommendations for the Development of Internet Network Management Standards [RFC-1052], a two-prong strategy for network management of TCP/IP-based internets was undertaken. In the short-term, the Simple Network Management Protocol (SNMP), defined in RFC-1067, was to be used to manage nodes in the Internet community. In the long-term, the use of the OSI network management framework was to be examined. Two documents were produced to define the management information: RFC-1065, which defined the Structure of Management Information (SMI), and RFC-1066, which defined the Management Information Base (MIB). Both of these documents were designed so as to be compatible with both the SNMP and the OSI network management framework.

This strategy was quite successful in the short-term: Internet-based network management technology was fielded, by both the research and commercial communities, within a few months. As a result of this, portions of the Internet community became network manageable in a timely fashion.

As reported in RFC-1109, Report of the Second Ad Hoc Network Management Review Group [RFC-1109], the requirements of the SNMP and the OSI network management frameworks were more different than anticipated. As such, the requirement for compatibility between the SMI/MIB and both frameworks was suspended. This action permitted the operational network management framework, based on the SNMP, to respond to new operational needs in the Internet community by producing MIB-II.

In May of 1990, the core documents were elevated to "Standard Protocols" with "Recommended" status. As such, the Internet-standard network management framework consists of: Structure and Identification of Management Information for TCP/IP-based internets, [RFC-1155], which describes how managed objects contained in the MIB are defined; Management Information Base for Network Management of TCP/IP-based internets, which describes the managed objects contained in the MIB, [RFC-1156]; and, the Simple Network Management Protocol, [RFC-1157], which defines the protocol used to manage these objects. Consistent with the IAB directive to produce simple, workable systems in the short-term, the list of managed objects defined in the Internet-standard MIB was derived by taking only those elements which are considered essential. However, the SMI defined three extensibility mechanisms: one, the addition of new standard objects through the definitions of new versions of the MIB; two, the addition of widely-available but non-standard objects through the experimental subtree; and three, the addition of private objects through the enterprises subtree. Such additional objects can not only be used for vendor-specific elements, but also for experimentation as required to further the knowledge of which other objects are essential.

As more objects are defined using the second method, experience has shown that the resulting MIB descriptions contain redundant information. In order to provide for MIB descriptions which are more concise, and yet as informative, an enhancement is suggested. This enhancement allows the author of a MIB to remove the redundant information, while retaining the important descriptive text.

Before presenting the approach, a brief presentation of columnar object handling by the SNMP is necessary. This explains and further motivates the value of the enhancement.

Columnar Objects

The SNMP supports operations on MIB objects whose syntax is ObjectSyntax as defined in the SMI. Informally stated, SNMP operations apply exclusively to scalar objects. However, it is convenient for developers of management applications to impose imaginary, tabular structures on the ordered collection of objects that constitute the MIB. Each such conceptual table contains zero or more rows, and each row may contain one or more scalar objects, termed columnar objects. Historically, this conceptualization has been formalized by using the OBJECT-TYPE macro to define both an object which corresponds to a table and an object which corresponds to a row in that table. (The ACCESS clause for such objects is "not-accessible", of course.) However, it must be emphasized that, at the protocol level, relationships among columnar objects in the same row is a matter of convention, not of protocol.

Note that there are good reasons why the tabular structure is not a matter of protocol. Consider the operation of the SNMP Get-Next-PDU acting on the last columnar object of an instance of a conceptual row; it returns the next column of the first conceptual row or the first object instance occurring after the table. In contrast, if the rows were a matter of protocol, then it would instead return an error. By not returning an error, a single PDU exchange informs the manager that not only has the end of the conceptual row/table been reached, but also provides information on the next object instance, thereby increasing the information density of the PDU exchange.

Row Deletion

Nonetheless, it is highly useful to provide a means whereby a conceptual row may be removed from a table. In MIB-II, this was achieved by defining, for each conceptual row, an integer-valued columnar object. If a management station sets the value of this object to some value, usually termed "invalid", then the effect is one of invalidating the corresponding row in the table. However, it is an implementation-specific matter as to whether an agent removes an invalidated entry from the table. Accordingly, management stations must be prepared to receive tabular information from agents that corresponds to entries not currently in use. Proper interpretation of such entries requires examination of the columnar object indicating the in-use status.

Row Addition

It is also highly useful to have a clear understanding of how a conceptual row may be added to a table. In the SNMP, at the protocol level, a management station issues an SNMP set operation containing an arbitrary set of variable bindings. In the case that an agent detects that one or more of those variable bindings refers to an object instance not currently available in that agent, it may, according to the rules of the SNMP, behave according to any of the following paradigms:

(1) It may reject the SNMP set operation as referring to non-existent object instances by returning a response with the error-status field set to "noSuchName" and the error-index field set to refer to the first vacuous reference.

(2) It may accept the SNMP set operation as requesting the creation of new object instances corresponding to each of the object instances named in the variable bindings. The value of each (potentially) newly created object instance is specified by the "value" component of the relevant variable binding. In this case, if the request specifies a value for a newly (or previously) created object that it deems inappropriate by reason of value or syntax, then it rejects the SNMP set operation by responding with the error-status field set to badValue and the error-index field set to refer to the first offending variable binding.

(3) It may accept the SNMP set operation and create new object instances as described in (2) above and, in addition, at its discretion, create supplemental object instances to complete a row in a conceptual table of which the new object instances specified in the request may be a part.

It should be emphasized that all three of the above behaviors are fully conformant to the SNMP specification and are fully acceptable, subject to any restrictions which may be imposed by access control and/or the definitions of the MIB objects themselves.

RFC-1212 Concise MIB Definitions

Defining Objects

The Internet-standard SMI employs a two-level approach towards object definition. A MIB definition consists of two parts: a textual part, in which objects are placed into groups, and a MIB module, in which objects are described solely in terms of the ASN.1 macro OBJECT-TYPE, which is defined by the SMI.

An example of the former definition might be:

```
OBJECT:
        sysLocation { system 6 }
Syntax:
        DisplayString (SIZE (0..255))
Definition:
        The physical location of this node (e.g., "telephone closet, 3rd
        floor").
Access:
        read-only.
Status:
        mandatory.
```

An example of the latter definition might be:

```
sysLocation OBJECT-TYPE
    SYNTAX   DisplayString (SIZE (0..255))
    ACCESS   read-only
    STATUS   mandatory
    ::= { system 6 }
```

In the interests of brevity and to reduce the chance of editing errors, it would seem useful to combine the two definitions. This can be accomplished by defining an extension to the OBJECT-TYPE macro:

```
IMPORTS
    ObjectName
FROM RFC1155-SMI
    DisplayString
FROM RFC1158-MIB;

OBJECT-TYPE MACRO ::=
BEGIN
    TYPE NOTATION ::=
            -- must conform to
            -- RFC1155's ObjectSyntax
        "SYNTAX" type(ObjectSyntax)
        "ACCESS" Access
        "STATUS" Status
        DescrPart
        ReferPart
        IndexPart
        DefValPart

    VALUE NOTATION ::= value (VALUE ObjectName)

    Access ::= "read-only"
```

```
        | "read-write"
        | "write-only"
        | "not-accessible"

    Status ::= "mandatory"
        | "optional"
        | "obsolete"
        | "deprecated"

    DescrPart ::=
        "DESCRIPTION" value (description DisplayString)
        | empty

    ReferPart ::=
        "REFERENCE" value (reference DisplayString)
        | empty

    IndexPart ::=
        "INDEX" "{" IndexTypes "}"
        | empty

    IndexTypes ::=
        IndexType | IndexTypes "," IndexType

    IndexType ::=
            -- if indexobject, use the SYNTAX
            -- value of the correspondent
            -- OBJECT-TYPE invocation
        value (indexobject ObjectName)
            -- otherwise use named SMI type
            -- must conform to IndexSyntax below
        | type (indextype)

    DefValPart ::=
        "DEFVAL" "{" value (defvalue ObjectSyntax) "}"
        | empty
END

IndexSyntax ::=
    CHOICE {
        number
            INTEGER (0..MAX),
        string
            OCTET STRING,
        object
            OBJECT IDENTIFIER,
        address
            NetworkAddress,
        ipAddress
            IpAddress
    }
```

Mapping of the OBJECT-TYPE macro

It should be noted that the expansion of the OBJECT-TYPE macro is something which conceptually happens during implementation and not during run-time.

Mapping of the SYNTAX clause

The SYNTAX clause, which must be present, defines the abstract data structure corresponding to that object type. The ASN.1 language [6] is used for this purpose. However, the SMI purposely restricts the ASN.1 constructs which may be used. These restrictions are made expressly for simplicity.

Mapping of the ACCESS clause

The ACCESS clause, which must be present, defines the minimum level of support required for that object type. As a local matter, implementations may support other access types (e.g., an implementation may elect to permitting writing a variable marked as read-only). Further, protocol-specific "views" (e.g., those indirectly implied by an SNMP community) may make further restrictions on access to a variable.

Mapping of the STATUS clause

The STATUS clause, which must be present, defines the implementation support required for that object type.

Mapping of the DESCRIPTION clause

The DESCRIPTION clause, which need not be present, contains a textual definition of that object type which provides all semantic definitions necessary for implementation, and should embody any information which would otherwise be communicated in any ASN.1 commentary annotations associated with the object. Note that, in order to conform to the ASN.1 syntax, the entire value of this clause must be enclosed in double quotation marks, although the value may be multi-line.

Further, note that if the MIB module does not contain a textual description of the object type elsewhere then the DESCRIPTION clause must be present.

Mapping of the REFERENCE clause

The REFERENCE clause, which need not be present, contains a textual cross-reference to an object defined in some other MIB module. This is useful when de-osifying a MIB produced by some other organization.

RFC-1212 Concise MIB Definitions: Mapping of the Object Type Macro

Mapping of the INDEX clause

The INDEX clause, which may be present only if that object type corresponds to a conceptual row, defines instance identification information for that object type. (Historically, each MIB definition contained a section entitled "Identification of OBJECT instances for use with the SNMP". By using the INDEX clause, this section need no longer occur as this clause concisely captures the precise semantics needed for instance identification.)

If the INDEX clause is not present, and the object type corresponds to a non-columnar object, then instances of the object are identified by appending a sub-identifier of zero to the name of that object. Further, note that if the MIB module does not contain a textual description of how instance identification information is derived for columnar objects, then the INDEX clause must be present.

To define the instance identification information, determine which object value(s) will unambiguously distinguish a conceptual row. The syntax of those objects indicate how to form the instance-identifier:

(1) integer-valued: a single sub-identifier taking the integer value (this works only for non-negative integers);

(2) string-valued, fixed-length strings: `n' sub-identifiers, where `n' is the length of the string (each octet of the string is encoded in a separate sub-identifier);

(3) string-valued, variable-length strings: `n+1' sub- identifiers, where `n' is the length of the string (the first sub-identifier is `n' itself, following this, each octet of the string is encoded in a separate sub- identifier);

(4) object identifier-valued: `n+1' sub-identifiers, where `n' is the number of sub-identifiers in the value (the first sub-identifier is `n' itself, following this, each sub-identifier in the value is copied);

(5) NetworkAddress-valued: `n+1' sub-identifiers, where `n' depends on the kind of address being encoded (the first sub-identifier indicates the kind of address, value 1 indicates an IpAddress); or,

(6) IpAddress-valued: 4 sub-identifiers, in the familiar a.b.c.d notation.

Note that if an "indextype" value is present (e.g., INTEGER rather than ifIndex), then a DESCRIPTION clause must be present; the text contained therein indicates the semantics of the "indextype" value.

By way of example, in the context of MIB-II [7], the following INDEX clauses might be present:

objects under	**INDEX clause**
ifEntry	{ ifIndex }
atEntry	{ atNetIfIndex, atNetAddress }
ipAddrEntry	{ ipAdEntAddr }
ipRouteEntry	{ ipRouteDest }
ipNetToMediaEntry	{ ipNetToMediaIfIndex, ipNetToMediaNetAddress }
	tcpConnEntry { tcpConnLocalAddress, tcpConnLocalPort, tcpConnRemoteAddress,

	tcpConnRemotePort }
udpEntry	{ udpLocalAddress, udpLocalPort }
egpNeighEntry	{ egpNeighAddr }

Mapping of the DEFVAL clause

The DEFVAL clause, which need not be present, defines an acceptable default value which may be used when an object instance is created at the discretion of the agent acting in conformance with the third paradigm described in Section 4.2 above.

During conceptual row creation, if an instance of a columnar object is not present as one of the operands in the correspondent SNMP set operation, then the value of the DEFVAL clause, if present, indicates an acceptable default value that the agent might use.

The value of the DEFVAL clause must, of course, correspond to the SYNTAX clause for the object. Note that if an operand to the SNMP set operation is an instance of a read-only object, then the error noSuchName will be returned. As such, the DEFVAL clause can be used to provide an acceptable default value that the agent might use.

It is possible that no acceptable default value may exist for any of the columnar objects in a conceptual row for which the creation of new object instances is allowed. In this case, the objects specified in the INDEX clause must have a corresponding ACCESS clause value of read-write.

By way of example, consider the following possible DEFVAL clauses:

ObjectSyntax	**DEFVAL clause**
INTEGER	1 -- same for Counter, Gauge, TimeTicks
OCTET STRING	'ffffffffffff'h
DisplayString	"any NVT ASCII string"
OBJECT IDENTIFIER	sysDescr
OBJECT IDENTIFIER	{ system 2 }
NULL	NULL
NetworkAddress	{ internet 'c0210415'h }
IpAddress	'c0210415'h -- 192.33.4.21

Mapping of the OBJECT-TYPE value

The value of an invocation of the OBJECT-TYPE macro is the name of the object, which is an object identifier.

Usage Example

Consider how the ipNetToMediaTable from MIB-II might be fully described:

```
        -- the IP Address Translation table

        -- The IP address translation table contain the IpAddress to
        -- `physical' address equivalences.   Some interfaces do not
        -- use translation tables for determining address
        -- equivalences (e.g., DDN-X.25 has an algorithmic method);
        -- if all interfaces are of this type, then the Address
        -- Translation table is empty, i.e., has zero entries.

        ipNetToMediaTable OBJECT-TYPE
                SYNTAX   SEQUENCE OF IpNetToMediaEntry
                ACCESS   not-accessible
                STATUS   mandatory
                DESCRIPTION
        "The IP Address Translation table used for mapping from IP addresses to
        physical addresses."
                ::= { ip 22 }

        ipNetToMediaEntry OBJECT-TYPE
                SYNTAX   IpNetToMediaEntry
                ACCESS   not-accessible
                STATUS   mandatory
                DESCRIPTION
        "Each entry contains one IpAddress to `physical' address equivalence."
                INDEX    { ipNetToMediaIfIndex,
                           ipNetToMediaNetAddress }
                ::= { ipNetToMediaTable 1 }

        IpNetToMediaEntry ::=
                SEQUENCE {
                    ipNetToMediaIfIndex
                        INTEGER,
                    ipNetToMediaPhysAddress
                        PhysAddress,
                    ipNetToMediaNetAddress
                        IpAddress,
                    ipNetToMediaType
                        INTEGER
                }

        ipNetToMediaIfIndex OBJECT-TYPE
                SYNTAX   INTEGER
                ACCESS   read-write
                STATUS   mandatory
                DESCRIPTION
        "The interface on which this entry's equivalence is effective.   The interface
        identified by a particular value of this index is the same interface as identified
```

by the same value of ifIndex."

::= { ipNetToMediaEntry 1 }

ipNetToMediaPhysAddress OBJECT-TYPE
SYNTAX PhysAddress
ACCESS read-write
STATUS mandatory
DESCRIPTION
"The media-dependent `physical' address."

::= { ipNetToMediaEntry 2 }

ipNetToMediaNetAddress OBJECT-TYPE
SYNTAX IpAddress
ACCESS read-write
STATUS mandatory
DESCRIPTION
"The IpAddress corresponding to the media- dependent `physical' address."

::= { ipNetToMediaEntry 3 }

ipNetToMediaType OBJECT-TYPE
SYNTAX INTEGER {
other(1), -- none of the following
invalid(2), -- an invalidated mapping
dynamic(3),
static(4)
}
ACCESS read-write
STATUS mandatory
DESCRIPTION
"The type of mapping.

Setting this object to the value invalid(2) has the effect of invalidating the corresponding entry in the ipNetToMediaTable. That is, it effectively dissasociates the interface identified with said entry from the mapping identified with said entry. It is an implementation-specific matter as to whether the agent removes an invalidated entry from the table. Accordingly, management stations must be prepared to receive tabular information from agents that corresponds to entries not currently in use. Proper interpretation of such entries requires examination of the relevant ipNetToMediaType object."

::= { ipNetToMediaEntry 4 }

Appendix: DE-osifying MIBs

There has been an increasing amount of work recently on taking MIBs defined by other organizations (e.g., the IEEE) and de-osifying them for use with the Internet-standard network management framework. The steps to achieve this are straight-forward, though tedious. Of course, it is helpful to already be experienced in writing MIB modules for use with the Internet-standard network management framework.

The first step is to construct a skeletal MIB module, e.g.,

```
RFC1213-MIB DEFINITIONS ::= BEGIN
IMPORTS
    experimental, OBJECT-TYPE, Counter
    FROM RFC1155-SMI;
                -- contact IANA for actual number
    root     OBJECT IDENTIFIER ::= { experimental xx }
END
```

The next step is to categorize the objects into groups. For experimental MIBs, optional objects are permitted. However, when a MIB module is placed in the Internet-standard space, these optional objects are either removed, or placed in a optional group, which, if implemented, all objects in the group must be implemented. For the first pass, it is wisest to simply ignore any optional objects in the original MIB: experience shows it is better to define a core MIB module first, containing only essential objects; later, if experience demands, other objects can be added.

It must be emphasized that groups are "units of conformance" within a MIB: everything in a group is "mandatory" and implementations do either whole groups or none.

Managed Object Mapping

Next for each managed object class, determine whether there can exist multiple instances of that managed object class. If not, then for each of its attributes, use the OBJECT-TYPE macro to make an equivalent definition.

Otherwise, if multiple instances of the managed object class can exist, then define a conceptual table having conceptual rows each containing a columnar object for each of the managed object class's attributes. If the managed object class is contained within the containment tree of another managed object class, then the assignment of an object type is normally required for each of the "distinguished attributes" of the containing managed object class. If they do not already exist within the MIB module, then they can be added via the definition of additional columnar objects in the conceptual row corresponding to the contained managed object class.

In defining a conceptual row, it is useful to consider the optimization of network management operations which will act upon its columnar objects. In particular, it is wisest to avoid defining more columnar objects within a conceptual row, than can fit in a single PDU. As a rule of thumb, a conceptual row should contain no more than approximately 20 objects. Similarly, or as a way to abide by the "20 object guideline", columnar objects should be grouped into tables according to the expected grouping of network management operations upon them. As such, the content of conceptual rows should reflect typical access scenarios, e.g., they should be organized along functional lines such as one row for statistics and another row for parameters, or along usage lines such as commonly-needed objects versus rarely-needed objects.

On the other hand, the definition of conceptual rows where the number of columnar objects used as indexes outnumbers the number used to hold information, should also be avoided. In particular, the splitting of a managed object class's attributes into many conceptual tables should not be used as a way to obtain the same degree of flexibility/complexity as is often found in MIB's with a myriad of optionals.

Mapping to the SYNTAX clause

When mapping to the SYNTAX clause of the OBJECT-type macro:

(1) An object with BOOLEAN syntax becomes an INTEGER taking either of values true(1) or false(2).

(2) An object with ENUMERATED syntax becomes an INTEGER, taking any of the values given.

(3) An object with BIT STRING syntax containing no more than 32 bits becomes an INTEGER defined as a sum; otherwise if more than 32 bits are present, the object becomes an OCTET STRING, with the bits numbered from left-to-right, in which the least significant bits of the last octet may be "reserved for future use".

(4) An object with a character string syntax becomes either an OCTET STRING or a DisplayString, depending on the repertoire of the character string.

(5) An non-tabular object with a complex syntax, such as REAL or EXTERNAL, must be decomposed, usually into an OCTET STRING (if sensible). As a rule, any object with a complicated syntax should be avoided.

(6) Tabular objects must be decomposed into rows of columnar objects.

Mapping to the ACCESS clause

This is straight-forward.

Mapping to the STATUS clause

This is usually straight-forward; however, some osified-MIBs use the term "recommended". In this case, a choice must be made between "mandatory" and "optional".

Mapping to the DESCRIPTION clause

This is straight-forward: simply copy the text, making sure that any embedded double quotation marks are sanitized (i.e., replaced with single-quotes or removed).

Mapping to the REFERENCE clause

This is straight-forward: simply include a textual reference to the object being mapped, the document which defines the object, and perhaps a page number in the document.

Mapping to the INDEX clause

Decide how instance-identifiers for columnar objects are to be formed and define this clause accordingly.

Mapping to the DEFVAL clause

Decide if a meaningful default value can be assigned to the object being mapped, and if so, define the DEFVAL clause accordingly.

Action Mapping

Actions are modeled as read-write objects, in which writing a particular value results in the action taking place.

Mapping to the SYNTAX clause

Usually an INTEGER syntax is used with a distinguished value provided for each action that the object provides access to. In addition, there is usually one other distinguished value, which is the one returned when the object is read.

Mapping to the ACCESS clause

Always use read-write.

Mapping to the STATUS clause

This is straight-forward.

Mapping to the DESCRIPTION clause

This is straight-forward: simply copy the text, making sure that any embedded double quotation marks are sanitized (i.e., replaced with single-quotes or removed).

Mapping to the REFERENCE clause

This is straight-forward: simply include a textual reference to the action being mapped, the document which defines the action, and perhaps a page number in the document.

Acknowledgements

This document was produced by the SNMP Working Group:

Anne Ambler, Spider
Karl Auerbach, Sun
Fred Baker, ACC
Ken Brinkerhoff
Ron Broersma, NOSC
Jack Brown, US Army
Theodore Brunner, Bellcore
Jeffrey Buffum, HP
John Burress, Wellfleet
Jeffrey D. Case, University of Tennessee at Knoxville
Chris Chiptasso, Spartacus
Paul Ciarfella, DEC
Bob Collet
John Cook, Chipcom
Tracy Cox, Bellcore
James R. Davin, MIT-LCS
Eric Decker, cisco
Kurt Dobbins, Cabletron
Nadya El-Afandi, Network Systems
Gary Ellis, HP
Fred Engle
Mike Erlinger
Mark S. Fedor, PSI
Richard Fox, Synoptics
Karen Frisa, CMU
Chris Gunner, DEC
Fred Harris, University of Tennessee at Knoxville
Ken Hibbard, Xylogics
Ole Jacobsen, Interop
Ken Jones
Satish Joshi, Synoptics
Frank Kastenholz, Racal-Interlan
Shimshon Kaufman, Spartacus
Ken Key, University of Tennessee at Knoxville
Jim Kinder, Fibercom
Alex Koifman, BBN
Christopher Kolb, PSI
Cheryl Krupczak, NCR
Paul Langille, DEC
Peter Lin, Vitalink
John Lunny, TWG
Carl Malamud
Randy Mayhew, University of Tennessee at Knoxville
Keith McCloghrie, Hughes LAN Systems
Donna McMaster, David Systems
Lynn Monsanto, Sun
Dave Perkins, 3COM
Jim Reinstedler, Ungerman Bass

Anil Rijsinghani, DEC
Kathy Rinehart, Arnold AFB
Kary Robertson
Marshall T. Rose, PSI (chair)
L. Michael Sabo, NCSC
Jon Saperia, DEC
Greg Satz, cisco
Martin Schoffstall, PSI
John Seligson
Steve Sherry, Xyplex
Fei Shu, NEC
Sam Sjogren, TGV
Mark Sleeper, Sparta
Lance Sprung
Mike St.Johns
Bob Stewart, Xyplex
Emil Sturniold
Kaj Tesink, Bellcore
Dean Throop, Data General
Bill Townsend, Xylogics
Maurice Turcotte, Racal-Milgo
Kannan Varadhou
Sudhanshu Verma, HP
Bill Versteeg, Network Research Corporation
Warren Vik, Interactive Systems
David Waitzman, BBN
Steve Waldbusser, CMU
Dan Wintringhan
David Wood
Wengyik Yeong, PSI
Jeff Young, Cray Research

Authors' Addresses

Keith McCloghrie
Hughes LAN Systems
1225 Charleston Road
Mountain View, CA 94043
1225 Charleston Road
Mountain View, CA 94043

Phone: (415) 966-7934
EMail: kzm@hls.com

Marshall T. Rose
Performance Systems International
5201 Great America Parkway
Suite 3106
Santa Clara, CA 95054

Phone: +1 408 562 6222
EMail: mrose@psi.com
X.500: rose, psi, us

7. References

[1] Cerf, V., "IAB Recommendations for the Development of Internet Network Management Standards", RFC 1052, NRI, April 1988.

[2] Cerf, V., "Report of the Second Ad Hoc Network Management Review Group", RFC 1109, NRI, August 1989.

[3] Rose M., and K. McCloghrie, "Structure and Identification of Management Information for TCP/IP-based internets", RFC 1155, Performance Systems International, Hughes LAN Systems, May 1990.

[4] McCloghrie K., and M. Rose, "Management Information Base for Network Management of TCP/IP-based internets", RFC 1156, Hughes LAN Systems, Performance Systems International, May 1990.

[5] Case, J., Fedor, M., Schoffstall, M., and J. Davin, "Simple Network Management Protocol", RFC 1157, SNMP Research, Performance Systems International, Performance Systems International, MIT Laboratory for Computer Science, May 1990.

[6] Information processing systems - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1), International Organization for Standardization International Standard 8824, December 1987.

[7] Rose M., Editor, "Management Information Base for Network Management of TCP/IP-based internets: MIB-II", RFC 1213, Performance Systems International, March 1991.

8. Security Considerations

Security issues are not discussed in this memo.

RFC-1213 Management Information Base
for
Network Managment of TCP/IP Based Internets
MIB-II

Keith McCloghrie & Marshall Rose; Editors
March 1991

Status of This Memo

This memo defines the second version of the Management Information Base (MIB-II) for use with network management protocols in TCP/IP- based internets. This RFC specifies an IAB standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "IAB Official Protocol Standards" for the standardization state and status of this protocol. Distribution of this memo is unlimited.

RFC-1213 Management Information Base: MIB-II

Abstract

This memo defines the second version of the Management Information Base (MIB-II) for use with network management protocols in TCP/IP- based internets. In particular, together with its companion memos which describe the structure of management information (RFC 1155) along with the network management protocol (RFC 1157) for TCP/IP- based internets, these documents provide a simple, workable architecture and system for managing TCP/IP-based internets and in particular the Internet community.

RFC-1213 Management Information Base: MIB-II

Introduction

As reported in IAB Recommendations for the Development of Internet Network Management Standards [RFC-1052], a two-prong strategy for network management of TCP/IP-based internets was undertaken. In the short-term, the Simple Network Management Protocol (SNMP) was to be used to manage nodes in the Internet community. In the long-term, the use of the OSI network management framework was to be examined. Two documents were produced to define the management information: The Structure of Management Information (SMI) [RFC-1155], and Management Information Base (MIB) [RFC-1156]. Both of these documents were designed so as to be compatible with both the SNMP and the OSI network management framework.

This strategy was quite successful in the short-term: Internet-based network management technology was fielded, by both the research and commercial communities, within a few months. As a result of this, portions of the Internet community became network manageable in a timely fashion.

As reported in Report of the Second Ad Hoc Network Management Review Group [RFC-1109], the requirements of the SNMP and the OSI network management frameworks were more different than anticipated. As such, the requirement for compatibility between the SMI/MIB and both frameworks was suspended. This action permitted the operational network management framework, the SNMP, to respond to new operational needs in the Internet community by producing this document.

As such, the current network management framework for TCP/IP- based internets consists of: Structure and Identification of Management Information for TCP/IP-based internets, [RFC-1155], which describes how managed objects contained in the MIB are defined; Management Information Base for Network Management of TCP/IP-based internets: MIB-II, this memo, which describes the managed objects contained in the MIB (and supercedes RFC-1156 Management Information Base[RFC-1156]); and, the Simple Network Management Protocol, [RFC-1157], which defines the protocol used to manage these objects.

RFC-1213 Management Information Base: MIB-II

Changes from RFC-1156

Although the current memo (RFC-1213) supercedes "Management Information Base" [RFC-1156], that memo is available is still listed as an IAB Standard and is included in the system.

Features of this MIB include:

(1) incremental additions to reflect new operational requirements;

(2) upwards compatibility with the SMI/MIB and the SNMP;

(3) improved support for multi-protocol entities; and,

(4) textual clean-up of the MIB to improve clarity and readability.

The objects defined in MIB-II have the OBJECT IDENTIFIER prefix:

```
mib-2        OBJECT IDENTIFIER ::= { mgmt 1 }
```

which is identical to the prefix used in MIB-I.

Depricated Objects
Display Strings
Physical Addresses
The Transmission Group
The SNMP Group

Deprecated Objects

In order to better prepare implementors for future changes in the MIB, a new term "deprecated" may be used when describing an object. A deprecated object in the MIB is one which must be supported, but one which will most likely be removed from the next version of the MIB (e.g., MIB-III).

MIB-II marks one object as being deprecated:

atTable

As a result of deprecating the atTable object, the entire Address Translation group is deprecated.

Note that no functionality is lost with the deprecation of these objects: new objects providing equivalent or superior functionality are defined in MIB-II.

Display Strings

In the past, there have been misinterpretations of the MIB as to when a string of octets should contain printable characters, meant to be displayed to a human. As a textual convention in the MIB, the datatype

```
DisplayString ::=
    OCTET STRING
```

is introduced. A DisplayString is restricted to the NVT ASCII character set, as defined in The Telnet Protocol [RFC-854].

The following objects are now defined in terms of DisplayString:

sysDescr
ifDescr

It should be noted that this change has no effect on either the syntax nor semantics of these objects. The use of the DisplayString notation is merely an artifact of the explanatory method used in MIB-II and future MIBs.

Further it should be noted that any object defined in terms of OCTET STRING may contain arbitrary binary data, in which each octet may take any value from 0 to 255 (decimal).

Physical Addresses

As a further, textual convention in the MIB, the datatype

```
PhysAddress ::=
    OCTET STRING
```

is introduced to represent media- or physical-level addresses.

The following objects are now defined in terms of PhysAddress:

ifPhysAddress
atPhysAddress
ipNetToMediaPhysAddress

It should be noted that this change has no effect on either the syntax nor semantics of these objects. The use of the PhysAddress notation is merely an artifact of the explanatory method used in MIB-II and future MIBs.

The Transmission Group

MIB-I was lacking in that it did not distinguish between different types of transmission media. A new group, the Transmission group, is allocated for this purpose:

transmission OBJECT IDENTIFIER ::= { mib-2 10 }

The SNMP Group

The application-oriented working groups of the IETF have been tasked to be receptive towards defining MIB variables specific to their respective applications.

For the SNMP, it is useful to have statistical information. A new group, the SNMP group, is allocated for this purpose:

```
snmp    OBJECT IDENTIFIER ::= { mib-2 11 }
```

Changes from RFC-1158

Since the current RFC (1213) obsoletes RFC-1158, RFC-1158 is not included in this system.

Features of this MIB include:

(1) The managed objects in this document have been defined using the conventions defined in the Internet-standard SMI, as amended by the extensions specified in [RFC-1212]. It must be emphasized that definitions made using these extensions are semantically identically to those in RFC-1158.

(2) The PhysAddress textual convention has been introduced to represent media addresses.

(3) The ACCESS clause of sysLocation is now read-write.

(4) The definition of sysServices has been clarified.

(5) New ifType values (29-32) have been defined. In addition, the textual-descriptor for the DS1 and E1 interface types has been corrected.

(6) The definition of ipForwarding has been clarified.

(7) The definition of ipRouteType has been clarified.

(8) The ipRouteMetric5 and ipRouteInfo objects have been defined.

(9) The ACCESS clause of tcpConnState is now read-write, to support deletion of the TCB associated with a TCP connection. The definition of this object has been clarified to explain this usage.

(10) The definition of egpNeighEventTrigger has been clarified.

(11) The definition of several of the variables in the new snmp group have been clarified. In addition, the snmpInBadTypes and snmpOutReadOnlys objects are no longer present. (However, the object identifiers associated with those objects are reserved to prevent future use.)

(12) The definition of snmpInReadOnlys has been clarified.

(13) The textual descriptor of the snmpEnableAuthTraps has been changed to snmpEnableAuthenTraps, and the definition has been clarified.

(14) The ipRoutingDiscards object was added.

(15) The optional use of an implementation-dependent, small positive integer was disallowed when identifying instances of the IP address and routing tables.

Objects

Managed objects are accessed via a virtual information store, termed the Management Information Base or MIB. Objects in the MIB are defined using the subset of Abstract Syntax Notation One (ASN.1) [8] defined in the SMI. In particular, each object has a name, a syntax, and an encoding. The name is an object identifier, an administratively assigned name, which specifies an object type. The object type together with an object instance serves to uniquely identify a specific instantiation of the object. For human convenience, we often use a textual string, termed the OBJECT DESCRIPTOR, to also refer to the object type.

The syntax of an object type defines the abstract data structure corresponding to that object type. The ASN.1 language is used for this purpose. However, the SMI [RFC-1155] purposely restricts the ASN.1 constructs which may be used. These restrictions are explicitly made for simplicity.

The encoding of an object type is simply how that object type is represented using the object type's syntax. Implicitly tied to the notion of an object type's syntax and encoding is how the object type is represented when being transmitted on the network.

The SMI specifies the use of the basic encoding rules of ASN.1 [9], subject to the additional requirements imposed by the SNMP.

Format of Definitions

The Object Definitions section contains the specification of all object types contained in this MIB module. The object types are defined using the conventions defined in the SMI, as amended by the extensions specified in [RFC-1212]. The format of these definitions is taken from MIB-I [RFC-1156].

Complete group definitions are also included to facilitate printing.

RFC-1213 Management Information Base: MIB-II

Overview

Consistent with the IAB directive to produce simple, workable systems in the short-term, the list of managed objects defined here, has been derived by taking only those elements which are considered essential.

This approach of taking only the essential objects is NOT restrictive, since the SMI defined in the companion memo provides three extensibility mechanisms: one, the addition of new standard objects through the definitions of new versions of the MIB; two, the addition of widely-available but non-standard objects through the experimental subtree; and three, the addition of private objects through the enterprises subtree. Such additional objects can not only be used for vendor-specific elements, but also for experimentation as required to further the knowledge of which other objects are essential.

The design of MIB-II is heavily influenced by the first extensibility mechanism. Several new variables have been added based on operational experience and need. Based on this, the criteria for including an object in MIB-II are remarkably similar to the MIB-I criteria:

(1) An object needed to be essential for either fault or configuration management.

(2) Only weak control objects were permitted (by weak, it is meant that tampering with them can do only limited damage). This criterion reflects the fact that the current management protocols are not sufficiently secure to do more powerful control operations.

(3) Evidence of current use and utility was required.

(4) In MIB-I, an attempt was made to limit the number of objects to about 100 to make it easier for vendors to fully instrument their software. In MIB-II, this limit was raised given the wide technological base now implementing MIB-I.

(5) To avoid redundant variables, it was required that no object be included that can be derived from others in the MIB.

(6) Implementation specific objects (e.g., for BSD UNIX) were excluded.

(7) It was agreed to avoid heavily instrumenting critical sections of code. The general guideline was one counter per critical section per layer.

MIB-II, like its predecessor, the Internet-standard MIB, contains only essential elements. There is no need to allow individual objects to be optional. Rather, the objects are arranged into the following groups:

- **System**
- **Interfaces**
- **Address Translation** (deprecated)
- **IP**
- **ICMP**
- **TCP**
- **UDP**
- **EGP**
- **Transmission**
- **SNMP**

These groups are the basic unit of conformance: This method is as follows: if the semantics of a group is applicable to an implementation, then it must implement all objects in that

group. For example, an implementation must implement the EGP group if and only if it implements the EGP.

There are two reasons for defining these groups: to provide a means of assigning object identifiers; and, to provide a method for implementations of managed agents to know which objects they must implement.

RFC-1213 Management Information Base for Network Management

Object Definitions

RFC-1213 Management Information Base for Network Management - Object Definitions

The System Group

Implementation of the System group is mandatory for all systems.

sysDescr
sysObjectID
sysUpTime
sysContact
sysName
sysLocation
sysServices

RFC-1213 Management Information Base for Network Management - System Group

OBJECT: **sysDescr** { system 1 }

Syntax:
DisplayString (SIZE (0..255))

Definition:
A textual description of the entity. This value should include the full name and version identification of the system's hardware type, software operating-system, and networking software. It is mandatory that this only contain printable ASCII characters.

Access:
read-only.

Status:
mandatory.

RFC-1213 Management Information Base for Network Management - System Group

OBJECT: **sysObjectID** { system 2 }

Syntax:
OBJECT IDENTIFIER

Definition:
The vendor's authoritative identification of the network management subsystem contained in the entity. This value is allocated within the SMI enterprises subtree (1.3.6.1.4.1) and provides an easy and unambiguous means for determining "what kind of box" is being managed. For example, if vendor "Flintstones, Inc." was assigned the subtree 1.3.6.1.4.1.42, it could assign the identifier 1.3.6.1.4.1.42.1.1 to its "Fred Router".

Access:
read-only.

Status:
mandatory.

RFC-1213 Management Information Base for Network Management - System Group

OBJECT: **sysUpTime** { system 3 }

Syntax:
TimeTicks

Definition:
The time (in hundredths of a second) since the network management portion of the system was last re-initialized.

Access:
read-only.

Status:
mandatory.

RFC-1213 Management Information Base for Network Management - System Group

OBJECT: **sysContact** { system 4 }

Syntax:
DisplayString (SIZE(0..255))

Definition:
The textual identification of the contact person for this managed node, together with information on how to contact this person.

Access:
read-write.

Status:
mandatory.

RFC-1213 Management Information Base for Network Management - System Group

OBJECT: **sysName** { system 5 }

Syntax:
DisplayString (SIZE(0..255))

Definition:
An administratively-assigned name for this managed node. By convention, this is the node's fully-qualified domain name.

Access:
read-write.

Status:
mandatory.

RFC-1213 Management Information Base for Network Management - System Group

OBJECT: **sysLocation** { system 6 }

Syntax:
DisplayString (SIZE(0..255))
Definition:
The physical location of this node (e.g., `telephone closet, 3rd floor').
Access:
read-write.
Status:
mandatory.

RFC-1213 Management Information Base for Network Management - System Group

OBJECT: **sysServices** { system 7 }

Syntax:

INTEGER (0..127)

Definition:

A value which indicates the set of services that this entity primarily offers.

The value is a sum. This sum initially takes the value zero, Then, for each layer, L, in the range 1 through 7, that this node performs transactions for, 2 raised to (L - 1) is added to the sum. For example, a node which performs primarily routing functions would have a value of 4 (2^(3-1)). In contrast, a node which is a host offering application services would have a value of 72 (2^(4-1) + 2^(7-1)). Note that in the context of the Internet suite of protocols, values should be calculated accordingly:

layer	functionality
1	physical (e.g., repeaters)
2	datalink/subnetwork (e.g., bridges)
3	internet (e.g., IP gateways)
4	end-to-end (e.g., IP hosts)
7	applications (e.g., mail relays)

For systems including OSI protocols, layers 5 and 6 may also be counted.

Access:

read-only.

Status:

mandatory.

The Interfaces Group

Implementation of the Interfaces group is mandatory for all systems. See also Interfaces Table.

OBJECT:
ifNumber { interfaces 1 }
Syntax:
INTEGER
Definition:
The number of network interfaces (regardless of their current state) present on this system.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base for Network Management - Object Definitions

The Interfaces Table

Administrative Objects:

ifTable
ifEntry
ifIndex
ifDescr
ifType
ifMTU
ifSpeed
ifPhysAddress
ifAdminStatus
ifOperStatus
ifLastChange

Input	Output
ifInOctets	**ifOutOctets**
ifInNUcastPkts	**ifOutUcastPkts**
ifInDiscards	**ifOutNUcastPkts**
ifInErrors	**ifOutDiscards**
ifInUnknownProtos	**ifOutQLen**

ifSpecific

RFC-1213 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifTable** { interfaces 2 }

Syntax:
SEQUENCE OF IfEntry

Definition:
A list of interface entries. The number of entries is given by the value of ifNumber.

Access:
not-accessible.

Status:
mandatory.

RFC-1213 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifEntry** { ifTable 1 }

Syntax:

```
IfEntry ::= SEQUENCE {
    ifIndex
        INTEGER,
    ifDescr
        DisplayString,
    ifType
        INTEGER,
    ifMtu
        INTEGER,
    ifSpeed
        Gauge,
    ifPhysAddress
        PhysAddress,
    ifAdminStatus
        INTEGER,
    ifOperStatus
        INTEGER,
    ifLastChange
        TimeTicks,
    ifInOctets
        Counter,
    ifInUcastPkts
        Counter,
    ifInNUcastPkts
        Counter,
    ifInDiscards
        Counter,
    ifInErrors
        Counter,
    ifInUnknownProtos
        Counter,
    ifOutOctets
        Counter,
    ifOutUcastPkts
        Counter,
    ifOutNUcastPkts
        Counter,
    ifOutDiscards
        Counter,
    ifOutErrors
        Counter,
    ifOutQLen
        Gauge
    ifSpecific
        OBJECT IDENTIFIER
    }
```

Definition:

An interface entry containing objects at the subnetwork layer and below for a

particular interface.

Access:

read-write.

Status:

mandatory.

RFC-1213 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifIndex** { ifEntry 1 }

Syntax:
INTEGER

Definition:
A unique value for each interface. Its value ranges between 1 and the value of ifNumber. The value for each interface must remain constant at least from one re- initialization of the entity's network management system to the next re-initialization.

Access:
read-only.

Status:
mandatory.

RFC-1213 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifDescr** { ifEntry 2 }

Syntax:
DisplayString (SIZE (0..255))

Definition:
A text string containing information about the interface. This string should include the name of the manufacturer, the product name and the version of the hardware interface. The string is intended for presentation to a human; it must not contain anything but printable ASCII characters.

Access:
read-only.

Status:
mandatory.

RFC-1213 Management Information Base for Network Management - Interfaces Table

OBJECT: ifType { ifEntry 3 }

Syntax:

```
INTEGER {
    other(1),               -- none of the following
    regular1822(2),
    hdh1822(3),
    ddn-x25(4),
    rfc877-x25(5),
    ethernet-csmacd(6),
    iso88023-csmacd(7),
    iso88024-tokenBus(8),
    iso88025-tokenRing(9),
    iso88026-man(10),
    starLan(11),
    proteon-10MBit(12),
    proteon-80MBit(13),
    hyperchannel(14),
    fddi(15),
    lapb(16),
    sdlc(17),
    t1-carrier(18),
    cept(19),               -- european equivalent of T-1
    basicIsdn(20),
    primaryIsdn(21),
                            -- proprietary serial
    propPointToPointSerial(22)
    ppp(23)
    softwareLoopback(24)
    eon(25)
    ethernet-3Mbit(26)
    nsip(27)
    slip(28)
    ultra(29)
    ds3(30)
    sip(31)
    frame-relay(32)
}
```

Definition:

The type of interface, distinguished according to the physical/link/network protocol(s) immediately "below" IP in the protocol stack.

Access:

read-only.

Status:

mandatory.

RFC-1213 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifMtu** { ifEntry 4 }

Syntax:
INTEGER
Definition:
The size of the largest IP datagram which can be sent/received on the interface, specified in octets.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifSpeed** { ifEntry 5 }

Syntax:
Gauge

Definition:
An estimate of the interface's current bandwidth in bits per second. For interfaces which do not vary in bandwidth or for those where no accurate estimation can be made, this object should contain the nominal bandwidth.

Access:
read-only.

Status:
mandatory.

RFC-1213 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifPhysAddress** { ifEntry 6 }

Syntax:
OCTET STRING

Definition:
The interface's address at the protocol layer immediately "below" IP in the protocol stack. For interfaces which do not have such an address (e.g., a serial line), this object should contain an octet string of zero length.

Access:
read-only.

Status:
mandatory.

RFC-1213 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifAdminStatus** { ifEntry 7 }

Syntax:

```
INTEGER {
    up(1),       -- ready to pass packets
    down(2),
    testing(3)  -- in some test mode
}
```

Definition:

The desired state of the interface. The testing(3) state indicates that no operational packets can be passed.

Access:

read-write.

Status:

mandatory.

RFC-1213 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifOperStatus** { ifEntry 8 }

Syntax:

```
INTEGER {
    up(1),          -- ready to pass packets
    down(2),
    testing(3)      -- in some test mode
}
```

Definition:

The current operational state of the interface. The testing(3) state indicates that no operational packets can be passed.

Access:

read-only.

Status:

mandatory.

RFC-1213 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifLastChange** { ifEntry 9 }

Syntax:
TimeTicks

Definition:
The value of sysUpTime at the time the interface entered its current operational state. If the current state was entered prior to the last re-initialization of the local network management subsystem, then this object contains a zero value.

Access:
read-only.

Status:
mandatory.

RFC-1213 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifInOctets** { ifEntry 10 }

Syntax:
Counter
Definition:
The total number of octets received on the interface, including framing characters.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifInUcastPkts** { ifEntry 11 }

Syntax:
Counter
Definition:
The number of (subnet) unicast packets delivered to a higher-layer protocol.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifInNUcastPkts** { ifEntry 12 }

Syntax:
Counter

Definition:
The number of non-unicast (i.e., subnet broadcast or subnet multicast) packets delivered to a higher-layer protocol.

Access:
read-only.

Status:
mandatory.

RFC-1213 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifInDiscards** { ifEntry 13 }

Syntax:
Counter

Definition:
The number of inbound packets which were chosen to be discarded even though no errors had been detected to prevent their being deliverable to a higher-layer protocol. One possible reason for discarding such a packet could be to free up buffer space.

Access:
read-only.

Status:
mandatory.

RFC-1213 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifInErrors** { ifEntry 14 }

Syntax:
Counter
Definition:
The number of inbound packets that contained errors preventing them from being deliverable to a higher-layer protocol.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifInUnknownProtos** { ifEntry 15 }

Syntax:
Counter

Definition:
The number of packets received via the interface which were discarded because of an unknown or unsupported protocol.

Access:
read-only.

Status:
mandatory.

RFC-1213 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifOutOctets** { ifEntry 16 }

Syntax:
Counter
Definition:
The total number of octets transmitted out of the interface, including framing characters.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifOutUcastPkts** { ifEntry 17 }

Syntax:
Counter

Definition:
The total number of packets that higher-level protocols requested be transmitted to a subnet-unicast address, including those that were discarded or not sent.

Access:
read-only.

Status:
mandatory.

RFC-1213 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifOutNUcastPkts** { ifEntry 18 }

Syntax:
Counter
Definition:
The total number of packets that higher-level protocols requested be transmitted to a non-unicast (i.e., a subnet broadcast or subnet multicast) address, including those that were discarded or not sent.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifOutDiscards** { ifEntry 19 }

Syntax:
Counter

Definition:
The number of outbound packets which were chosen to be discarded even though no errors had been detected to prevent their being transmitted. One possible reason for discarding such a packet could be to free up buffer space.

Access:
read-only.

Status:
mandatory.

RFC-1213 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifOutErrors** { ifEntry 20 }

Syntax:
Counter
Definition:
The number of outbound packets that could not be transmitted because of errors.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifOutQLen** { ifEntry 21 }

Syntax:
Gauge
Definition:
The length of the output packet queue (in packets).
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base for Network Management - Interfaces Table

OBJECT: **ifSpecific** { ifEntry 22 }

Syntax:
OBJECT IDENTIFIER

Definition:
A reference to MIB definitions specific to the particular media being used to realize the interface. For example, if the interface is realized by an ethernet, then the value of this object refers to a document defining objects specific to ethernet. If this information is not present, its value should be set to the OBJECT IDENTIFIER { 0 0 }, which is a syntatically valid object identifier, and any conformant implementation of ASN.1 and BER must be able to generate and recognize this value.

Access:
read-only.

Status:
mandatory.

The Address Translation Group

In MIB-I [RFC-1156] this group contained a table which permitted mappings from network addresses (e.g., IP addresses) to physical addresses (e.g., MAC addresses). Experience has shown that efficient implementations of this table make two assumptions: a single network protocol environment, and mappings occur only from network address to physical address.

The need to support multi-protocol nodes (e.g., those with both the IP and CLNP active), and the need to support the inverse mapping (e.g., for ES-IS), have invalidated both of these assumptions. As such, the atTable object is declared deprecated. It is documented here for historical reference.

In order to meet both the multi-protocol and inverse mapping requirements, MIB-II and its successors will allocate up to two address translation tables inside each network protocol group. That is, the IP group will contain one address translation table, for going from IP addresses to physical addresses. Similarly, when a document defining MIB objects for the CLNP is produced (e.g., [RFC-1162]), it will contain two tables, for mappings in both directions, as this is required for full functionality.

It should be noted that the choice of two tables (one for each direction of mapping) provides for ease of implementation in many cases, and does not introduce undue burden on implementations which realize the address translation abstraction through a single internal table.

The Address Translation group contains one table which is the union across all interfaces of the translation tables for converting a NetworkAddress (e.g., an IP address) into a subnetwork-specific address. For lack of a better term, this document refers to such a subnetwork-specific address as a "physical" address.

Examples of such translation tables are: for broadcast media where ARP is in use, the translation table is equivalent to the ARP cache; or, on an X.25 network where non-algorithmic translation to X.121 addresses is required, the translation table contains the NetworkAddress to X.121 address equivalences.

atTable
atEntry
atIfIndex
atPhysAddress
atNetAddress

RFC-1213 Management Information Base - Address Translation Group

OBJECT: **atTable** { at 1 }

Syntax:
SEQUENCE OF AtEntry

Definition:
The Address Translation tables contain the NetworkAddress to "physical" address equivalences. Some interfaces do not use translation tables for determining address equivalences (e.g., DDN-X.25 has an algorithmic method); if all interfaces are of this type, then the Address Translation table is empty, i.e., has zero entries.

Access:
read-write.

Status:
mandatory.

RFC-1213 Management Information Base - Address Translation Group

OBJECT: **atEntry** { atTable 1 }

Syntax:

```
AtEntry ::= SEQUENCE {
    atIfIndex
        INTEGER,
    atPhysAddress
        PhysAddress,
    atNetAddress
        NetworkAddress
}
```

Definition:

Each entry contains one NetworkAddress to "physical" address equivalence.

Access:

read-write.

Status:

mandatory.

RFC-1213 Management Information Base - Address Translation Group

OBJECT: **atIfIndex** { atEntry 1 }

Syntax:
INTEGER
Definition:
The interface on which this entry's equivalence is effective. The interface identified by a particular value of this index is the same interface as identified by the same value of ifIndex.
Access:
read-write.
Status:
mandatory.

RFC-1213 Management Information Base - Address Translation Group

OBJECT: **atPhysAddress** { atEntry 2 }

Syntax:
OCTET STRING
Definition:
The media-dependent "physical" address.
Access:
read-write.
Status:
mandatory.

RFC-1213 Management Information Base - Address Translation Group

OBJECT: **atNetAddress** { atEntry 3 }

Syntax:
NetworkAddress
Definition:
The NetworkAddress (e.g., the IP address) corresponding to the media-dependent "physical" address.
Access:
read-write.
Status:
mandatory.

The IP Group

Implementation of the IP group is mandatory for all systems.

ipForwarding
ipDefaultTTL
ipInReceives
ipInHdrErrors
ipInAddrErrors
ipForwDatagrams
ipInUnknownProtos
ipInDiscards
ipInDelivers
ipOutRequests
ipOutDiscards
ipOutNoRoutes
ipReasmTimeout
ipReasmReqds
ipReasmOKs
ipReasmFails
ipFragOKs
ipFragFails
ipFragCreates

ipAddressTable

ipRouteTable

ipAddressTranslationTable

ipNetToMediaTable

ipRoutingDiscards

RFC-1213 Management Information Base - IP Group

OBJECT: **ipForwarding** { ip 1 }

Syntax:

```
INTEGER {
    gateway(1),         -- entity forwards datagrams
    host(2)     -- entity does NOT forward datagrams
}
```

Definition:

The indication of whether this entity is acting as an IP gateway in respect to the forwarding of datagrams received by, but not addressed to, this entity. IP gateways forward datagrams; Hosts do not (except those Source-Routed via the host).

Access:

read-write.

Status:

mandatory.

RFC-1213 Management Information Base - IP Group

OBJECT: **ipDefaultTTL** { ip 2 }

Syntax:
INTEGER

Definition:
The default value inserted into the Time-To-Live field of the IP header of datagrams originated at this entity, whenever a TTL value is not supplied by the transport layer protocol.

Access:
read-write.

Status:
mandatory.

RFC-1213 Management Information Base - IP Group

OBJECT: **ipInReceives** { ip 3 }

Syntax:
Counter
Definition:
The total number of input datagrams received from interfaces, including those received in error.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - IP Group

OBJECT: **ipInHdrErrors** { ip 4 }

Syntax:
Counter
Definition:
The number of input datagrams discarded due to errors in their IP headers, including bad checksums, version number mismatch, other format errors, time-to-live exceeded, errors discovered in processing their IP options, etc.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - IP Group

OBJECT: **ipInAddrErrors** { ip 5 }

Syntax:
Counter

Definition:
The number of input datagrams discarded because the IP address in their IP header's destination field was not a valid address to be received at this entity. This count includes invalid addresses (e.g., 0.0.0.0) and addresses of unsupported Classes (e.g., Class E). For entities which are not IP Gateways and therefore do not forward datagrams, this counter includes datagrams discarded because the destination address was not a local address.

Access:
read-only.

Status:
mandatory.

RFC-1213 Management Information Base - IP Group

OBJECT: **ipForwDatagrams** { ip 6 }

Syntax:
Counter

Definition:
The number of input datagrams for which this entity was not their final IP destination, as a result of which an attempt was made to find a route to forward them to that final destination. In entities which do not act as IP Gateways, this counter will include only those packets which were Source-Routed via this entity, and the Source-Route option processing was successful.

Access:
read-only.

Status:
mandatory.

RFC-1213 Management Information Base - IP Group

OBJECT: **ipInUnknownProtos** { ip 7 }

Syntax:
Counter
Definition:
The number of locally-addressed datagrams received successfully but discarded because of an unknown or unsupported protocol.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - IP Group

OBJECT: **ipInDiscards** { ip 8 }

Syntax:
Counter

Definition:
The number of input IP datagrams for which no problems were encountered to prevent their continued processing, but which were discarded (e.g. for lack of buffer space). Note that this counter does not include any datagrams discarded while awaiting re-assembly.

Access:
read-only.

Status:
mandatory.

RFC-1213 Management Information Base - IP Group

OBJECT: **ipInDelivers** { ip 9 }

Syntax:
Counter
Definition:
The total number of input datagrams successfully delivered to IP user-protocols (including ICMP).
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - IP Group

OBJECT: **ipOutRequests** { ip 10 }

Syntax:
Counter

Definition:
The total number of IP datagrams which local IP user- protocols (including ICMP) supplied to IP in requests for transmission. Note that this counter does not include any datagrams counted in ipForwDatagrams.

Access:
read-only.

Status:
mandatory.

RFC-1213 Management Information Base - IP Group

OBJECT: **ipOutDiscards** { ip 11 }

Syntax:
Counter

Definition:
The number of output IP datagrams for which no problem was encountered to prevent their transmission to their destination, but which were discarded (e.g., for lack of buffer space). Note that this counter would include datagrams counted in ipForwDatagrams if any such packets met this (discretionary) discard criterion.

Access:
read-only.

Status:
mandatory.

RFC-1213 Management Information Base - IP Group

OBJECT: **ipOutNoRoutes** { ip 12 }

Syntax:
Counter
Definition:
The number of IP datagrams discarded because no route could be found to transmit them to their destination. Note that this counter includes any packets counted in ipForwDatagrams which meet this "no-route" criterion.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - IP Group

OBJECT: **ipReasmTimeout** { ip 13 }

Syntax:
INTEGER
Definition:
The maximum number of seconds which received fragments are held while they are awaiting reassembly at this entity.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - IP Group

OBJECT: **ipReasmReqds** { ip 14 }

Syntax:
Counter
Definition:
The number of IP fragments received which needed to be reassembled at this entity.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - IP Group

OBJECT: **ipReasmOKs** { ip 15 }

Syntax:
Counter
Definition:
The number of IP datagrams successfully re-assembled.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - IP Group

OBJECT: **ipReasmFails** { ip 16 }

Syntax:
Counter

Definition:
The number of failures detected by the IP re-assembly algorithm (for whatever reason: timed out, errors, etc).
Note that this is not necessarily a count of discarded IP fragments since some algorithms (notably RFC 815's) can lose track of the number of fragments by combining them as they are received.

Access:
read-only.

Status:
mandatory.

RFC-1213 Management Information Base - IP Group

OBJECT: **ipFragOKs** { ip 17 }

Syntax:
Counter
Definition:
The number of IP datagrams that have been successfully fragmented at this entity.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - IP Group

OBJECT: **ipFragFails** { ip 18 }

Syntax:
Counter

Definition:
The number of IP datagrams that have been discarded because they needed to be fragmented at this entity but could not be, e.g., because their "Don't Fragment" flag was set.

Access:
read-only.

Status:
mandatory.

RFC-1213 Management Information Base - IP Group

OBJECT: **ipFragCreates** { ip 19 }

Syntax:
Counter
Definition:
The number of IP datagram fragments that have been generated as a result of fragmentation at this entity.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base for Network Management - IP Group

The IP Address Table

The Ip Address table contains this entity's IP addressing information.

ipAddrTable
ipAddrEntry
ipAdEntAddr
ipAdEntIfIndex
ipAdEntNetMask
ipAdEntBcastAddr
ipAdEntReasmMaxSize

RFC-1213 Management Information Base - IP Address Table

OBJECT: **ipAddrTable** { ip 20 }

Syntax:
SEQUENCE OF IpAddrEntry
Definition:
The table of addressing information relevant to this entity's IP addresses.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - IP Address Table

OBJECT: **ipAddrEntry** { ipAddrTable 1 }

Syntax:

```
IpAddrEntry ::= SEQUENCE {
    ipAdEntAddr
        IpAddress,
    ipAdEntIfIndex
        INTEGER,
    ipAdEntNetMask
        IpAddress,
    ipAdEntBcastAddr
        INTEGER
    ipAdEntReasmMaxSize
        INTEGER (0..65535)
}
```

Definition:

The addressing information for one of this entity's IP addresses.

Access:

read-only.

Status:

mandatory.

RFC-1213 Management Information Base - IP Address Table

OBJECT: **ipAdEntAddr** { ipAddrEntry 1 }

Syntax:
IpAddress
Definition:
The IP address to which this entry's addressing information pertains.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - IP Address Table

OBJECT: **ipAdEntIfIndex** { ipAddrEntry 2 }

Syntax:
INTEGER
Definition:
The index value which uniquely identifies the interface to which this entry is applicable. The interface identified by a particular value of this index is the same interface as identified by the same value of ifIndex.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - IP Address Table

OBJECT: **ipAdEntNetMask** { ipAddrEntry 3 }

Syntax:
IpAddress
Definition:
The subnet mask associated with the IP address of this entry. The value of the mask is an IP address with all the network bits set to 1 and all the hosts bits set to 0.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - IP Address Table

OBJECT: **ipAdEntBcastAddr** { ipAddrEntry 4 }

Syntax:
INTEGER
Definition:
The value of the least-significant bit in the IP broadcast address used for sending datagrams on the (logical) interface associated with the IP address of this entry. For example, when the Internet standard all-ones broadcast address is used, the value will be 1.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - IP Address Table

OBJECT: **ipAdEntReasmMaxSize** { ipAddrEntry 5 }

Syntax:
INTEGER (0..65535)
Definition:
The size of the largest IP datagram which this entity can re-assemble from incoming IP fragmented datagrams received on this interface.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base for Network Management - IP Group

The IP Route Table

The IP Routing Table contains an entry for each route presently known to this entity. Note that the action to be taken in response to a request to read a non-existent entry, is specific to the network management protocol being used.

ipRouteTable
ipRouteEntry
ipRouteDest
ipRouteIfIndex
ipRouteMetric1
ipRouteMetric2
ipRouteMetric3
ipRouteMetric4
ipRouteMetric5
ipRouteNextHop
ipRouteType
ipRouteProto
ipRouteAge
ipRouteMask
ipRouteInfo

RFC-1213 Management Information Base - IP Routing Table

OBJECT: **ipRouteTable** { ip 21 }

Syntax:
SEQUENCE OF IpRouteEntry
Definition:
This entity's IP Routing table.
Access:
read-write.
Status:
mandatory.

RFC-1213 Management Information Base - IP Routing Table

OBJECT: **ipRouteEntry** { ipRouteTable 1 }

Syntax:

```
IpRouteEntry ::= SEQUENCE {
    ipRouteDest
        IpAddress,
    ipRouteIfIndex
        INTEGER,
    ipRouteMetric1
        INTEGER,
    ipRouteMetric2
        INTEGER,
    ipRouteMetric3
        INTEGER,
    ipRouteMetric4
        INTEGER,
    ipRouteNextHop
        IpAddress,
    ipRouteType
        INTEGER,
    ipRouteProto
        INTEGER,
    ipRouteAge
        INTEGER
    ipRouteMask
        IpAddress,
    ipRouteMetric5
        INTEGER,
    ipRouteInfo
        OBJECT IDENTIFIER
}
```

Definition:

A route to a particular destination.

Access:

read-write.

Status:

mandatory.

RFC-1213 Management Information Base - IP Routing Table

OBJECT: **ipRouteDest** { ipRouteEntry 1 }

Syntax:
IpAddress
Definition:
The destination IP address of this route. An entry with a value of 0.0.0.0 is considered a default route. Multiple such default routes can appear in the table, but access to such multiple entries is dependent on the table-access mechanisms defined by the network management protocol in use.
Access:
read-write.
Status:
mandatory.

RFC-1213 Management Information Base - IP Routing Table

OBJECT: **ipRouteIfIndex** { ipRouteEntry 2 }

Syntax:
INTEGER
Definition:
The index value which uniquely identifies the local interface through which the next hop of this route should be reached. The interface identified by a particular value of this index is the same interface as identified by the same value of ifIndex.
Access:
read-write.
Status:
mandatory.

RFC-1213 Management Information Base - IP Routing Table

OBJECT: **ipRouteMetric1** { ipRouteEntry 3 }

Syntax:
INTEGER
Definition:
The primary routing metric for this route. The semantics of this metric are determined by the routing-protocol specified in the route's ipRouteProto value. If this metric is not used, its value should be set to -1.
Access:
read-write.
Status:
mandatory.

RFC-1213 Management Information Base - IP Routing Table

OBJECT: **ipRouteMetric2** { ipRouteEntry 4 }

Syntax:
INTEGER
Definition:
An alternate routing metric for this route. The semantics of this metric are determined by the routing- protocol specified in the route's ipRouteProto value. If this metric is not used, its value should be set to -1.
Access:
read-write.
Status:
mandatory.

RFC-1213 Management Information Base - IP Routing Table

OBJECT: **ipRouteMetric3** { ipRouteEntry 5 }

Syntax:
INTEGER
Definition:
An alternate routing metric for this route. The semantics of this metric are determined by the routing- protocol specified in the route's ipRouteProto value. If this metric is not used, its value should be set to -1.
Access:
read-write.
Status:
mandatory.

RFC-1213 Management Information Base - IP Routing Table

OBJECT: **ipRouteMetric4** { ipRouteEntry 6 }

Syntax:
INTEGER
Definition:
An alternate routing metric for this route. The semantics of this metric are determined by the routing- protocol specified in the route's ipRouteProto value. If this metric is not used, its value should be set to -1.
Access:
read-write.
Status:
mandatory.

RFC-1213 Management Information Base - IP Routing Table

OBJECT: **ipRouteMetric5** { ipRouteEntry 12 }

Syntax:
INTEGER
Definition:
An alternate routing metric for this route. The semantics of this metric are determined by the routing-protocol specified in the route's ipRouteProto value. If this metric is not used, its value should be set to -1.
Access:
read-write.
Status:
mandatory.

RFC-1213 Management Information Base - IP Routing Table

OBJECT: **ipRouteNextHop** { ipRouteEntry 7 }

Syntax:
IpAddress
Definition:
The IP address of the next hop of this route.
Access:
read-write.
Status:
mandatory.

RFC-1213 Management Information Base - IP Routing Table

OBJECT: **ipRouteType** { ipRouteEntry 8 }

Syntax:

```
INTEGER {
    other(1),   -- none of the following
    invalid(2), -- an invalidated route
    direct(3),  -- route to directly connected (sub-)network
    remote(4),-- route to a non-local host/network/sub-network
}
```

Definition:

The type of route.

Access:

read-write.

Status:

mandatory.

RFC-1213 Management Information Base - IP Routing Table

OBJECT: **ipRouteProto** { ipRouteEntry 9 }

Syntax:

```
INTEGER {
    other(1),       -- none of the following

                    -- non-protocol information,
                    -- e.g., manually configured
    local(2),       -- entries

                    -- set via a network management
    netmgmt(3),         -- protocol

                    -- obtained via ICMP,
    icmp(4),        -- e.g., Redirect

                    -- the remaining values are
                    -- all gateway routing protocols
    egp(5),
    ggp(6),
    hello(7),
    rip(8),
    is-is(9),
    es-is(10),
    ciscoIgrp(11),
    bbnSpfIgp(12),
    oigp(13)
}
```

Definition:

The routing mechanism via which this route was learned. Inclusion of values for gateway routing protocols is not intended to imply that hosts should support those protocols.

Access:

read-only.

Status:

mandatory.

RFC-1213 Management Information Base - IP Routing Table

OBJECT: **ipRouteAge** { ipRouteEntry 10 }

Syntax:
INTEGER
Definition:
The number of seconds since this route was last updated or otherwise determined to be correct. Note that no semantics of "too old" can be implied except through knowledge of the routing protocol by which the route was learned.
Access:
read-write.
Status:
mandatory.

RFC-1213 Management Information Base - IP Routing Table

OBJECT: **ipRouteMask** { ipRouteEntry 11 }

Syntax:
ipAddress

Definition:
Indicate the mask to be logical-ANDed with the destination address before being compared to the value in the ipRouteDest field. For those systems that do not support arbitrary subnet masks, an agent constructs the value of the ipRouteMask by determining whether the value of the correspondent ipRouteDest field belong to a class-A, B, or C network, and then using one of:

mask	**network**
255.0.0.0	class-A
255.255.0.0	class-B
255.255.255.0	class-C

If the value of the ipRouteDest is 0.0.0.0 (a default route), then the mask value is also 0.0.0.0. It should be noted that all IP routing subsystems implicitly use this mechanism.

Access:
read-write.

Status:
mandatory.

RFC-1213 Management Information Base - IP Routing Table

OBJECT: **ipRouteInfo** { ipRouteEntry 13 }

Syntax:
INTEGER
Definition:
A reference to MIB definitions specific to the particular routing protocol which is responsible for this route, as determined by the value specified in the route's ipRouteProto value. If this information is not present, its value should be set to the OBJECT IDENTIFIER { 0 0 }, which is a syntatically valid object identifier, and any conformant implementation of ASN.1 and BER must be able to generate and recognize this value.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - IP Routing Table

The IP Address Translation Table

The IP address translation table contain the IpAddress to `physical' address equivalences. Some interfaces do not use translation tables for determining address equivalences (e.g., DDN-X.25 has an algorithmic method); if all interfaces are of this type, then the Address Translation table is empty, i.e., has zero entries.

- **ipNetToMediaTable**
  - **ipNetToMediaEntry**
    - **ipNetToMediaIfIndex**
    - **ipNetToMediaNetAddress**
      - **ipNetToMediaPhysAddress**
      - **ipNetToMediaNetAddress**
    - **ipNetToMediaType**

RFC-1213 Management Information Base - IP Address Translation Table

OBJECT: **ipNetToMediaTable** { ip 22 }

Syntax:
SEQUENCE OF lpNetToMediaEntry
Definition:
The IP Address Translation table used for mapping from IP addresses to physical addresses.
Access:
not-accessible
Status:
mandatory.

RFC-1213 Management Information Base - IP Address Translation Table

OBJECT: **ipNetToMediaEntry** { ipNetToMediaTable 1 }

Syntax:

IpNetToMediaEntry ::=
SEQUENCE {
ipNetToMediaIfIndex
INTEGER,
ipNetToMediaPhysAddress
PhysAddress,
ipNetToMediaNetAddress
IpAddress,
ipNetToMediaType
INTEGER
}

Definition:

Each entry contains one IpAddress to `physical' address equivalence.

Access:

not-accessible

Status:

mandatory.

RFC-1213 Management Information Base - IP Address Translation Table

OBJECT: **ipNetToMediaIfIndex** { ipNetToMediaEntry 1 }

Syntax:
INTEGER
Definition:
The interface on which this entry's equivalence is effective. The interface identified by a particular value of this index is the same interface as identified by the same value of ifIndex.
Access:
read-write
Status:
mandatory.

RFC-1213 Management Information Base - IP Address Translation Table

OBJECT: **ipNetToMediaPhysAddress** { ipNetToMediaEntry 2 }

Syntax:
PhysAddress
Definition:
The media-dependent `physical' address.
Access:
read-write
Status:
mandatory.

RFC-1213 Management Information Base - IP Address Translation Table

OBJECT: **ipNetToMediaNetAddress** { ipNetToMediaEntry 3}

Syntax:
IpAddress
Definition:
The IpAddress corresponding to the media- dependent `physical' address.
Access:
read-write
Status:
mandatory.

RFC-1213 Management Information Base - IP Address Translation Table

OBJECT: **ipNetToMediaType** { ipNetToMediaEntry 4}

Syntax:

INTEGER {
other(1) -- none of the following
invalid(2) -- an invalidated mapping
dynamic(3)
static(4)
}

Definition:

The type of mapping.

Setting this object to the value invalid(2) has the effect of invalidating the corresponding entry in the ipNetToMediaTable. That is, it effectively dissasociates the interface identified with said entry. It is an implementation-specific matter as to whether the agent removes an invalidated entry from the table. Accordingly, management stations must be prepared to receive tabular information from agents that corresponds to entries not currently in use. Proper interpretation of such entries requires examination of the relevant ipNetToMediaType object.

Access:

read-write

Status:

mandatory.

RFC-1213 Management Information Base - IP Group

OBJECT: **ipRoutingDiscards** { ip 23}

Syntax:
Counter
Definition:
The number of routing entries which were chosen to be discarded even though they are valid. One possible reason for discarding such an entry could be to free-up buffer space for other routing entries.
Access:
read-only
Status:
mandatory.

The ICMP Group

Implementation of the ICMP group is mandatory for all systems.

The ICMP group contains the ICMP input and output statistics.

Note that individual counters for ICMP message (sub-)codes have been omitted from this (version of the) MIB for simplicity.

icmpInMsgs	**icmpOutMsgs**
icmpInErrors	**icmpOutErrors**
icmpInDestUnreachs	**icmpOutDestUnreachs**
icmpInTimeExcds	**icmpOutTimeExcds**
icmpInParmProbs	**icmpOutParmProbs**
icmpInSrcQuenchs	**icmpOutSrcQuenchs**
icmpInRedirects	**icmpOutRedirects**
icmpInEchos	**icmpOutEchos**
icmpInEchoReps	**icmpOutEchoReps**
icmpInTimestamps	**icmpOutTimestamps**
icmpInTimestampReps	**icmpOutTimestampReps**
icmpInAddrMasks	**icmpOutAddrMasks**
icmpInAddrMaskReps	**icmpOutAddrMaskReps**

RFC-1213 Management Information Base - ICMP Group

OBJECT: **icmpInMsgs** { icmp 1 }

Syntax:
Counter
Definition:
The total number of ICMP messages which the entity received. Note that this counter includes all those counted by icmpInErrors.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - ICMP Group

OBJECT: **icmpInErrors** { icmp 2 }

Syntax:
Counter
Definition:
The number of ICMP messages which the entity received but determined as having errors (bad ICMP checksums, bad length, etc.).
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - ICMP Group

OBJECT: **icmpInDestUnreachs** { icmp 3 }

Syntax:
Counter
Definition:
The number of ICMP Destination Unreachable messages received.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - ICMP Group

OBJECT: **icmpInTimeExcds** { icmp 4 }

Syntax:
Counter
Definition:
The number of ICMP Time Exceeded messages received.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - ICMP Group

OBJECT: **icmpInParmProbs** { icmp 5 }

Syntax:
Counter
Definition:
The number of ICMP Parameter Problem messages received.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - ICMP Group

OBJECT: **icmpInSrcQuenchs** { icmp 6 }

Syntax:
Counter
Definition:
The number of ICMP Source Quench messages received.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - ICMP Group

OBJECT: **icmpInRedirects** { icmp 7 }

Syntax:
Counter
Definition:
The number of ICMP Redirect messages received.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - ICMP Group

OBJECT: **icmpInEchos** { icmp 8 }

Syntax:
Counter
Definition:
The number of ICMP Echo (request) messages received.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - ICMP Group

OBJECT: **icmpInEchoReps** { icmp 9 }

Syntax:
Counter
Definition:
The number of ICMP Echo Reply messages received.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - ICMP Group

OBJECT: **icmpInTimestamps** { icmp 10 }

Syntax:
Counter
Definition:
The number of ICMP Timestamp (request) messages received.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - ICMP Group

OBJECT: **icmpInTimestampReps** { icmp 11 }

Syntax:
Counter
Definition:
The number of ICMP Timestamp Reply messages received.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - ICMP Group

OBJECT: **icmpInAddrMasks** { icmp 12 }

Syntax:
Counter
Definition:
The number of ICMP Address Mask Request messages received.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - ICMP Group

OBJECT: **icmpInAddrMaskReps** { icmp 13 }

Syntax:
Counter
Definition:
The number of ICMP Address Mask Reply messages received.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - ICMP Group

OBJECT: **icmpOutMsgs** { icmp 14 }

Syntax:
Counter
Definition:
The total number of ICMP messages which this entity attempted to send. Note that this counter includes all those counted by icmpOutErrors.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - ICMP Group

OBJECT: **icmpOutErrors** { icmp 15 }

Syntax:
Counter

Definition:
The number of ICMP messages which this entity did not send due to problems discovered within ICMP such as a lack of buffers. This value should not include errors discovered outside the ICMP layer such as the inability of IP to route the resultant datagram. In some implementations there may be no types of error which contribute to this counter's value.

Access:
read-only.

Status:
mandatory.

RFC-1213 Management Information Base - ICMP Group

OBJECT: **icmpOutDestUnreachs** { icmp 16 }

Syntax:
Counter
Definition:
The number of ICMP Destination Unreachable messages sent.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - ICMP Group

OBJECT: **icmpOutTimeExcds** { icmp 17 }

Syntax:
Counter
Definition:
The number of ICMP Time Exceeded messages sent.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - ICMP Group

OBJECT: **icmpOutParmProbs** { icmp 18 }

Syntax:
Counter
Definition:
The number of ICMP Parameter Problem messages sent.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - ICMP Group

OBJECT: **icmpOutSrcQuenchs** { icmp 19 }

Syntax:
Counter
Definition:
The number of ICMP Source Quench messages sent.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - ICMP Group

OBJECT: **icmpOutRedirects** { icmp 20 }

Syntax:
Counter
Definition:
The number of ICMP Redirect messages sent.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - ICMP Group

OBJECT: **icmpOutEchos** { icmp 21 }

Syntax:
Counter
Definition:
The number of ICMP Echo (request) messages sent.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - ICMP Group

OBJECT: **icmpOutEchoReps** { icmp 22 }

Syntax:
Counter
Definition:
The number of ICMP Echo Reply messages sent.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - ICMP Group

OBJECT: **icmpOutTimestamps** { icmp 23 }

Syntax:
Counter
Definition:
The number of ICMP Timestamp (request) messages sent.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - ICMP Group

OBJECT: **icmpOutTimestampReps** { icmp 24 }

Syntax:
Counter
Definition:
The number of ICMP Timestamp Reply messages sent.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - ICMP Group

OBJECT: **icmpOutAddrMasks** { icmp 25 }

Syntax:
Counter
Definition:
The number of ICMP Address Mask Request messages sent.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - ICMP Group

OBJECT: **icmpOutAddrMaskReps** { icmp 26 }

Syntax:
Counter
Definition:
The number of ICMP Address Mask Reply messages sent.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base for Network Management - Object Definitions

The TCP Group

Implementation of the TCP group is mandatory for all systems that implement the TCP protocol.

Note that instances of object types that represent information about a particular TCP connection are transient; they persist only as long as the connection in question.

RFC-1213 Management Information Base - TCP Group

OBJECT: **tcpRtoAlgorithm** { tcp 1 }

Syntax:

```
INTEGER {
    other(1),      -- none of the following
    constant(2), -- a constant rto
    rsre(3),       -- MIL-STD-1778, Appendix B
    vanj(4)        -- Van Jacobson's algorithm [15]
}
```

Definition:

The algorithm used to determine the timeout value used for retransmitting unacknowledged octets.

Access:

read-only.

Status:

mandatory.

RFC-1213 Management Information Base - TCP Group

OBJECT: **tcpRtoMin** { tcp 2 }

Syntax:
INTEGER
Definition:
The minimum value permitted by a TCP implementation for the retransmission timeout, measured in milliseconds. More refined semantics for objects of this type depend upon the algorithm used to determine the retransmission timeout. In particular, when the timeout algorithm is rsre(3), an object of this type has the semantics of the LBOUND quantity described in RFC 793.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - TCP Group

OBJECT: **tcpRtoMax** { tcp 3 }

Syntax:
INTEGER

Definition:
The maximum value permitted by a TCP implementation for the retransmission timeout, measured in milliseconds. More refined semantics for objects of this type depend upon the algorithm used to determine the retransmission timeout. In particular, when the timeout algorithm is rsre(3), an object of this type has the semantics of the UBOUND quantity described in RFC 793.

Access:
read-only.

Status:
mandatory.

RFC-1213 Management Information Base - TCP Group

OBJECT: **tcpMaxConn** { tcp 4 }

Syntax:
INTEGER
Definition:
The limit on the total number of TCP connections the entity can support. In entities where the maximum number of connections is dynamic, this object should contain the value "-1".
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - TCP Group

OBJECT: **tcpActiveOpens** { tcp 5 }

Syntax:
Counter
Definition:
The number of times TCP connections have made a direct transition to the SYN-SENT state from the CLOSED state.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - TCP Group

OBJECT: **tcpPassiveOpens** { tcp 6 }

Syntax:
Counter
Definition:
The number of times TCP connections have made a direct transition to the SYN-RCVD state from the LISTEN state.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - TCP Group

OBJECT: **tcpAttemptFails** { tcp 7 }

Syntax:
Counter
Definition:
The number of times TCP connections have made a direct transition to the CLOSED state from either the SYN-SENT state or the SYN-RCVD state, plus the number of times TCP connections have made a direct transition to the LISTEN state from the SYN-RCVD state.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - TCP Group

OBJECT: **tcpEstabResets** { tcp 8 }

Syntax:
Counter
Definition:
The number of times TCP connections have made a direct transition to the CLOSED state from either the ESTABLISHED state or the CLOSE-WAIT state.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - TCP Group

OBJECT: **tcpCurrEstab** { tcp 9 }

Syntax:
Gauge
Definition:
The number of TCP connections for which the current state is either ESTABLISHED or CLOSE-WAIT.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - TCP Group

OBJECT: **tcpInSegs** { tcp 10 }

Syntax:
Counter
Definition:
The total number of segments received, including those received in error.
This count includes segments received on currently established connections.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - TCP Group

OBJECT: **tcpOutSegs** { tcp 11 }

Syntax:
Counter
Definition:
The total number of segments sent, including those on current connections but excluding those containing only retransmitted octets.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - TCP Group

OBJECT: **tcpRetransSegs** { tcp 12 }

Syntax:
Counter
Definition:
The total number of segments retransmitted - that is, the number of TCP segments transmitted containing one or more previously transmitted octets.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - TCP Group

OBJECT: **tcpConnTable** { tcp 13 }

Syntax:
SEQUENCE OF TcpConnEntry
Definition:
A table containing TCP connection-specific information.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - TCP Group

OBJECT: **tcpConnEntry** { tcpConnTable 1 }

Syntax:

```
TcpConnEntry ::= SEQUENCE {
    tcpConnState
        INTEGER,
    tcpConnLocalAddress
        IpAddress,
    tcpConnLocalPort
        INTEGER (0..65535),
    tcpConnRemAddress
        IpAddress,
    tcpConnRemPort
        INTEGER (0..65535)
}
```

Definition:

Information about a particular current TCP connection. An object of this type is transient, in that it ceases to exist when (or soon after) the connection makes the transition to the CLOSED state.

Access:

read-only.

Status:

mandatory.

RFC-1213 Management Information Base - TCP Group

OBJECT: **tcpConnState** { tcpConnEntry 1 }

Syntax:

INTEGER {
closed(1),
listen(2),
synSent(3),
synReceived(4),
established(5),
finWait1(6),
finWait2(7),
closeWait(8),
lastAck(9),
closing(10),
timeWait(11)
}

Definition:

The state of this TCP connection.

Access:

read-only.

Status:

mandatory.

RFC-1213 Management Information Base - TCP Group

OBJECT: **tcpConnLocalAddress** { tcpConnEntry 2 }

Syntax:
IpAddress
Definition:
The local IP address for this TCP connection.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - TCP Group

OBJECT: **tcpConnLocalPort** { tcpConnEntry 3 }

Syntax:
INTEGER (0..65535**)**
Definition:
The local port number for this TCP connection.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - TCP Group

OBJECT: **tcpConnRemAddress** { tcpConnEntry 4 }

Syntax:
IpAddress
Definition:
The remote IP address for this TCP connection.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - TCP Group

OBJECT: **tcpConnRemPort** { tcpConnEntry 5 }

Syntax:
INTEGER (0..65535)
Definition:
The remote port number for this TCP connection.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - TCP Group

OBJECT: **tcpInErrs** { tcp 14 }

Syntax:
Counter
Definition:
The total number of segments received in error (e.g., bad TCP checksums).
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - TCP Group

OBJECT: **tcpOutRsts** { tcp 15 }

Syntax:
Counter
Definition:
The number of TCP segments sent containing the RST flag.
Access:
read-only.
Status:
mandatory.

The UDP Group

Implementation of the UDP group is mandatory for all systems which implement the UDP protocol.

udpInDatagrams
udpNoPorts
udpInErrors
udpOutDatagrams

udpTable
- **udpEntry**
  - **udpLocalAddress**
  - **udpLocalPort**

RFC-1213 Management Information Base - UDP Group

OBJECT: **udpInDatagrams** { udp 1 }

Syntax:
Counter
Definition:
The total number of UDP datagrams delivered to UDP users.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - UDP Group

OBJECT: **udpNoPorts** { udp 2 }

Syntax:
Counter
Definition:
The total number of received UDP datagrams for which there was no application at the destination port.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - UDP Group

OBJECT: **udpInErrors** { udp 3 }

Syntax:
Counter
Definition:
The number of received UDP datagrams that could not be delivered for reasons other than the lack of an application at the destination port.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - UDP Group

OBJECT: **udpOutDatagrams** { udp 4 }

Syntax:
Counter
Definition:
The total number of UDP datagrams sent from this entity.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - UDP Group

OBJECT: **udpTable** { udp 5 }

The UDP listener table contains information about this entity's UDP end-points on which a local application is currently accepting datagrams.

Syntax:
SEQUENCE OF udpEntry
Definition:
A table containing UDP listener information.
Access:
not-accessible
Status:
mandatory.

RFC-1213 Management Information Base - UDP Table

OBJECT: **udpEntry** { udpTable 1 }

Syntax:

SEQUENCE {
udpLocalAddress
IpAddress,
udpLocalPort
INTEGER (0..65535)
}

Definition:

Information about a particular current UDP listener.

Access:

not-accessible

Status:

mandatory.

RFC-1213 Management Information Base - UDP Table

OBJECT: **udpLocalAddress** { udpEntry 1 }

Syntax:
IpAddress
Definition:
The local IP address for this UDP listener. In the case of a UDP listener which is willing to accept datagrams for any IP interface associated with the node, the value 0.0.0.0 is used.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - UDP Table

OBJECT: **udpLocalPort** { udpEntry 2 }

Syntax:
INTEGER (0..65535)
Definition:
The local port number for this UDP listener.
Access:
read-only.
Status:
mandatory.

The EGP Group

Implementation of the EGP group is mandatory for all systems which implement the EGP protocol.

egpInMsgs
egpInErrors
egpOutMsgs
egpOutErrors
egpAs

RFC-1213 Management Information Base - EGP Group

OBJECT: **egpInMsgs** { egp 1 }

Syntax:
Counter
Definition:
The number of EGP messages received without error.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - EGP Group

OBJECT: **egpInErrors** { egp 2 }

Syntax:
Counter
Definition:
The number of EGP messages received that proved to be in error.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - EGP Group

OBJECT: **egpOutMsgs** { egp 3 }

Syntax:
Counter
Definition:
The total number of locally generated EGP messages.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - EGP Group

OBJECT: **egpOutErrors** { egp 4 }

Syntax:
Counter
Definition:
The number of locally generated EGP messages not sent due to resource limitations within an EGP entity.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - EGP Group

OBJECT: **egpAs** { egp 6 }

Syntax:
INTEGER
Definition:
The autonomous system number of this EGP entity.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base for Network Management - Object Definitions

The EGP Neighbor Table

The Egp Neighbor table contains information about this entity's EGP neighbors.

egpNeighTable
egpNeighEntry
egpNeighState
egpNeighAddr
egpNeighAs
egpNeighInMsgs
egpNeighInErrs
egpNeighOutMsgs
egpNeighOutErrs
egpNeighInErrMsgs
egpNeighOutErrMsgs
egpNeighStateUps
egpNeighStateDowns
egpNeighIntervalHello
egpNeighIntervalPoll
egpNeighMode
egpNeighEventTrigger

RFC-1213 Management Information Base - EGP Neighbor Table

OBJECT: **egpNeighTable** { egp 5 }

Syntax:
SEQUENCE OF EgpNeighEntry
Definition:
The EGP neighbor table.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - EGP Neighbor Table

OBJECT: **egpNeighEntry** { egpNeighTable 1 }

Syntax:

SEQUENCE {

egpNeighState
INTEGER,
egpNeighAddr
IpAddress,
egpNeighAs
INTEGER,
egpNeighInMsgs
Counter,
egpNeighInErrs
Counter,
egpNeighOutMsgs
Counter,
egpNeighOutErrs
Counter,
egpNeighInErrMsgs
Counter,
egpNeighOutErrMsgs
Counter,
egpNeighStateUps
Counter,
egpNeighStateDowns
Counter,
egpNeighIntervalHello
INTEGER,
egpNeighIntervalPoll
INTEGER,
egpNeighMode
INTEGER,
egpNeighEventTrigger
INTEGER

}

Definition:

Information about this entity's relationship with a particular EGP neighbor.

Access:

read-only.

Status:

mandatory.

RFC-1213 Management Information Base - EGP Neighbor Table

OBJECT: **egpNeighState** { egpNeighEntry 1 }

Syntax:

INTEGER {
idle(1),
acquisition(2),
down(3),
up(4),
cease(5)
}

Definition:

The EGP state of the local system with respect to this entry's EGP neighbor. Each EGP state is represented by a value that is one greater than the numerical value associated with said state in RFC 904.

Access:

read-only.

Status:

mandatory.

RFC-1213 Management Information Base - EGP Neighbor Table

OBJECT: **egpNeighAddr** { egpNeighEntry 2 }

Syntax:
IpAddress
Definition:
The IP address of this entry's EGP neighbor.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - EGP Neighbor Group

OBJECT: **egpNeighAs** { egpNeighEntry 3 }

Syntax:
INTEGER
Definition:
The autonomous system of this EGP peer. Zero should be specified if the autonomous system number of the neighbor is not yet known.
Access:
read-only
Status:
mandatory.

RFC-1213 Management Information Base - EGP Neighbor Group

OBJECT: **egpNeighInMsgs** { egpNeighEntry 4 }

Syntax:
Counter
Definition:
The number of EGP messages received without error from this EGP peer.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - EGP Neighbor Group

OBJECT: **egpNeighInErrs** { egpNeighEntry 5 }

Syntax:
Counter
Definition:
The number of EGP messages received from this EGP peer that proved to be in error (e.g., bad EGP checksum).
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - EGP Neighbor Group

OBJECT: **egpNeighOutMsgs** { egpNeighEntry 6 }

Syntax:
Counter
Definition:
The number of locally generated EGP messages to this EGP peer.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - EGP Neighbor Group

OBJECT: **egpNeighOutErrs** { egpNeighEntry 7 }

Syntax:
Counter
Definition:
The number of locally generated EGP messages not sent to this EGP peer due to resource limitations within an EGP entity.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - EGP Neighbor Group

OBJECT: **egpNeighInErrMsgs** { egpNeighEntry 8 }

Syntax:
Counter
Definition:
The number of EGP-defined error messages received from this EGP peer.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - EGP Neighbor Group

OBJECT: **egpNeighOutErrMsgs** { egpNeighEntry 9 }

Syntax:
Counter
Definition:
The number of EGP-defined error messages sent to this EGP peer.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - EGP Neighbor Group

OBJECT: **egpNeighStateUps** { egpNeighEntry 10 }

Syntax:
Counter
Definition:
The number of EGP state transitions to the UP state with this EGP peer.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - EGP Neighbor Group

OBJECT: **egpNeighStateDowns** { egpNeighEntry 11 }

Syntax:
Counter
Definition:
The number of EGP state transitions from the UP state to any other state with this EGP peer.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - EGP Neighbor Group

OBJECT: **egpNeighIntervalHello** { egpNeighEntry 12 }

Syntax:
INTEGER
Definition:
The interval between EGP Hello command retransmissions (in hundredths of a second). This represents the t1 timer as defined in RFC 904.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - EGP Neighbor Group

OBJECT: **egpNeighIntervalPoll** { egpNeighEntry 13 }

Syntax:
INTEGER
Definition:
The interval between EGP poll command retransmissions (in hundredths of a second). This represents the t3 timer as defined in RFC 904.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - EGP Neighbor Group

OBJECT: **egpNeighMode** { egpNeighEntry 14 }

Syntax:
INTEGER { active(1), passive(2) }
Definition:
The polling mode of this EGP entity, either passive or active.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - EGP Neighbor Group

OBJECT: **egpNeighEventTrigger** { egpNeighEntry 15 }

Syntax:
INTEGER { start(1), stop(2) }

Definition:
A control variable used to trigger operator- initiated Start and Stop events. When read, this variable always returns the most recent value that egpNeighEventTrigger was set to. If it has not been set since the last initialization of the network management subsystem on the node, it returns a value of `stop'.

When set, this variable causes a Start or Stop event on the specified neighbor, as specified on pages 8-10 of RFC 904. Briefly, a Start event causes an Idle peer to begin neighbor acquisition and a non-Idle peer to reinitiate neighbor acquisition. A stop event causes a non-Idle peer to return to the Idle state until a Start event occurs, either via egpNeighEventTrigger or otherwise.

Access:
read-write.

Status:
mandatory.

The Transmission Group

Based on the transmission media underlying each interface on a system, the corresponding portion of the Transmission group is mandatory for that system.

When Internet-standard definitions for managing transmission media are defined, the transmission group is used to provide a prefix for the names of those objects.

Typically, such definitions reside in the experimental portion of the MIB until they are "proven", then as a part of the Internet standardization process, the definitions are accordingly elevated and a new object identifier, under the transmission group is defined. By convention, the name assigned is:

```
type OBJECT IDENTIFIER      ::= { transmission number }
```

where "type" is the symbolic value used for the media in the ifType column of the ifTable object,and "number" is the actual integer value corresponding to the symbol.

RFC-1213 Management Information Base - Object Definitions

The SNMP Group

Implementation of the SNMP group is mandatory for all systems which support an SNMP protocol entity. Some of the objects defined below will be zero-valued in those SNMP implementations that are optimized to support only those functions specific to either a management agent or a management station. In particular, it should be observed that the objects below refer to an SNMP entity, and there may be several SNMP entities residing on a managed node (e.g., if the node is hosting acting as a management station).

snmpInPkts	**snmpOutPkts**
snmpInBadVersions	**snmpInBadCommunityNames**
snmpInBadCommunityUses	**snmpInASNParseErrs**
snmpInTooBigs	**snmpInNoSuchNames**
snmpInBadValues	**snmpInReadOnlys**
snmpInGenErrs	**snmpInTotalReqVars**
snmpInTotalSetVars	**snmpInGetRequests**
snmpInGetNexts	**snmpInSetRequests**
snmpInGetResponses	**snmpInTraps**
snmpOutTooBigs	**snmpOutNoSuchNames**
snmpOutBadValues	**snmpOutGenErrs**
snmpOutGetRequests	**snmpOutGetNexts**
snmpOutSetRequests	**snmpOutGetResponses**
snmpOutTraps	**snmpEnableAuthenTraps**

RFC-1213 Management Information Base - SNMP Group

OBJECT: **snmpInPkts** { snmp 1 }

Syntax:
Counter
Definition:
The total number of Messages delivered to the SNMP entity from the transport service.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - SNMP Group

OBJECT: **snmpOutPkts** { snmp 2 }

Syntax:
Counter
Definition:
The total number of SNMP Messages which were passed from the SNMP protocol entity to the transport service.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - SNMP Group

OBJECT: **snmpInBadVersions** { snmp 3 }

Syntax:
Counter
Definition:
The total number of SNMP Messages which were delivered to the SNMP protocol entity and were for an unsupported SNMP version.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - SNMP Group

OBJECT: **snmpInBadCommunityNames** { snmp 4 }

Syntax:
Counter
Definition:
The total number of SNMP Messages delivered to the SNMP protocol entity which used a SNMP community name not known to said entity.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - SNMP Group

OBJECT: **snmpInBadCommunityUses** { snmp 5 }

Syntax:
Counter
Definition:
The total number of SNMP Messages delivered to the SNMP protocol entity which represented an SNMP operation which was not allowed by the SNMP community named in the Message.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - SNMP Group

OBJECT: **snmpInASNParseErrs** { snmp 6 }

Syntax:
Counter
Definition:
The total number of ASN.1 or BER errors encountered by the SNMP protocol entity when decoding received SNMP Messages.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - SNMP Group

OBJECT: **snmpInTooBigs** { snmp 8 }

Syntax:
Counter
Definition:
The total number of SNMP PDUs which were delivered to the SNMP protocol entity and for which the value of the error-status field is `tooBig'.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - SNMP Group

OBJECT: **snmpInNoSuchNames** { snmp 9 }

Syntax:
Counter
Definition:
The total number of SNMP PDUs which were delivered to the SNMP protocol entity and for which the value of the error-status field is `noSuchName'.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - SNMP Group

OBJECT: **snmpInBadValues** { snmp 10 }

Syntax:
Counter
Definition:
The total number of SNMP PDUs which were delivered to the SNMP protocol entity and for which the value of the error-status field is `badValue'.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - SNMP Group

OBJECT: **snmpInReadOnlys** { snmp 11 }

Syntax:
Counter

Definition:
The total number valid SNMP PDUs which were delivered to the SNMP protocol entity and for which the value of the error-status field is `readOnly'. It should be noted that it is a protocol error to generate an SNMP PDU which contains the value `readOnly' in the error-status field, as such this object is provided as a means of detecting incorrect implementations of the SNMP.

Access:
read-only.

Status:
mandatory.

RFC-1213 Management Information Base - SNMP Group

OBJECT: **snmpInGenErrs** { snmp 12 }

Syntax:
Counter
Definition:
The total number of SNMP PDUs which were delivered to the SNMP protocol entity and for which the value of the error-status field is `genErr'.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - SNMP Group

OBJECT: **snmpInTotalReqVars** { snmp 13 }

Syntax:
Counter
Definition:
The total number of MIB objects which have been retrieved successfully by the SNMP protocol entity as the result of receiving valid SNMP Get-Request and Get-Next PDUs.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - SNMP Group

OBJECT: **snmpInTotalSetVars** { snmp 14 }

Syntax:
Counter
Definition:
The total number of MIB objects which have been altered successfully by the SNMP protocol entity as the result of receiving valid SNMP Set-Request PDUs.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - SNMP Group

OBJECT: **snmpInGetRequests** { snmp 15 }

Syntax:
Counter
Definition:
The total number of SNMP Get-Request PDUs which have been accepted and processed by the SNMP protocol entity.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - SNMP Group

OBJECT: **snmpInGetNexts** { snmp 16 }

Syntax:
Counter
Definition:
The total number of SNMP Get-Next PDUs which have been accepted and processed by the SNMP protocol entity.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - SNMP Group

OBJECT: **snmpInSetRequests** { snmp 17 }

Syntax:
Counter
Definition:
The total number of SNMP Set-Request PDUs which have been accepted and processed by the SNMP protocol entity.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - SNMP Group

OBJECT: **snmpInGetResponses** { snmp 18 }

Syntax:
Counter
Definition:
The total number of SNMP Get-Response PDUs which have been accepted and processed by the SNMP protocol entity.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - SNMP Group

OBJECT: **snmpInTraps** { snmp 19 }

Syntax:
Counter
Definition:
The total number of SNMP Trap PDUs which have been accepted and processed by the SNMP protocol entity.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - SNMP Group

OBJECT: **snmpOutTooBigs** { snmp 20 }

Syntax:
Counter
Definition:
The total number of SNMP PDUs which were generated by the SNMP protocol entity and for which the value of the error-status field is `tooBig.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - SNMP Group

OBJECT: **snmpOutNoSuchNames** { snmp 21 }

Syntax:
Counter
Definition:
The total number of SNMP PDUs which were generated by the SNMP protocol entity and for which the value of the error-status is `noSuchName'.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - SNMP Group

OBJECT: **snmpOutBadValues** { snmp 22 }

Syntax:
Counter
Definition:
The total number of SNMP PDUs which were generated by the SNMP protocol entity and for which the value of the error-status field is `badValue'.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - SNMP Group

OBJECT: **snmpOutGenErrs** { snmp 24 }

Syntax:
Counter
Definition:
The total number of SNMP PDUs which were generated by the SNMP protocol entity and for which the value of the error-status field is `genErr'.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - SNMP Group

OBJECT: **snmpOutGetRequests** { snmp 25 }

Syntax:
Counter
Definition:
The total number of SNMP Get-Request PDUs which have been generated by the SNMP protocol entity.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - SNMP Group

OBJECT: **snmpOutGetNexts** { snmp 26 }

Syntax:
Counter
Definition:
The total number of SNMP Get-Next PDUs which have been generated by the SNMP protocol entity.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - SNMP Group

OBJECT: **snmpOutSetRequests** { snmp 27 }

Syntax:
Counter
Definition:
The total number of SNMP Set-Request PDUs which have been generated by the SNMP protocol entity.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - SNMP Group

OBJECT: **snmpOutGetResponses** { snmp 28 }

Syntax:
Counter
Definition:
The total number of SNMP Get-Response PDUs which have been generated by the SNMP protocol entity.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - SNMP Group

OBJECT: **snmpOutTraps** { snmp 29 }

Syntax:
Counter
Definition:
The total number of SNMP Trap PDUs which have been generated by the SNMP protocol entity.
Access:
read-only.
Status:
mandatory.

RFC-1213 Management Information Base - SNMP Group

OBJECT: **snmpEnableAuthenTraps** { snmp 30 }

Syntax:
INTEGER { enabled(1), disabled(2) }

Definition:
Indicates whether the SNMP agent process is permitted to generate authentication-failure traps. The value of this object overrides any configuration information; as such, it provides a means whereby all authentication-failure traps may be disabled.

Note that it is strongly recommended that this object be stored in non-volatile memory so that it remains constant between re-initializations of the network management system.

Access:
read-write.

Status:
mandatory.

RFC-1213 Management Information Base for Network Management

Group Definitions

The definitions in this section are identical to those in the preceeding sections, but are arranged so that an entire group is contained in a single "topic" and may therefore be printed out.

Groups in MIB-II

```
system          OBJECT IDENTIFIER ::= { mib-2 1 }
interfaces      OBJECT IDENTIFIER ::= { mib-2 2 }
at              OBJECT IDENTIFIER ::= { mib-2 3 }
ip              OBJECT IDENTIFIER ::= { mib-2 4 }
icmp            OBJECT IDENTIFIER ::= { mib-2 5 }
tcp             OBJECT IDENTIFIER ::= { mib-2 6 }
udp             OBJECT IDENTIFIER ::= { mib-2 7 }
egp             OBJECT IDENTIFIER ::= { mib-2 8 }
-- historical (some say hysterical)
-- cmot         OBJECT IDENTIFIER ::= { mib-2 9 }
transmission    OBJECT IDENTIFIER ::= { mib-2 10 }
snmp            OBJECT IDENTIFIER ::= { mib-2 11 }

RFC1213-MIB DEFINITIONS ::= BEGIN

IMPORTS
        mgmt, NetworkAddress, IpAddress, Counter, Gauge,
                TimeTicks
            FROM RFC1155-SMI
        OBJECT-TYPE
                FROM RFC-1212;

--  This MIB module uses the extended OBJECT-TYPE macro as
--  defined in [RFC-1212];

--  MIB-II (same prefix as MIB-I)

mib-2       OBJECT IDENTIFIER ::= { mgmt 1 }

-- textual conventions

DisplayString ::=
        OCTET STRING
-- This data type is used to model textual information taken
-- from the NVT ASCII character set.   By convention, objects
-- with this syntax are declared as having
--
--      SIZE (0..255)

PhysAddress ::=
        OCTET STRING
-- This data type is used to model media addresses.   For many
-- types of media, this will be in a binary representation.
-- For example, an ethernet address would be represented as
```

-- a string of 6 octets.

The System Group

```
-- Implementation of the System group is mandatory for all
-- systems.   If an agent is not configured to have a value
-- for any of these variables, a string of length 0 is
-- returned.

sysDescr OBJECT-TYPE
        SYNTAX   DisplayString (SIZE (0..255))
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION
"A textual description of the entity.   This value should include the full name
and version identification of the system's hardware type, software operating-
system, and networking software.   It is mandatory that this only contain
printable ASCII characters."

        ::= { system 1 }

sysObjectID OBJECT-TYPE
        SYNTAX   OBJECT IDENTIFIER
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION
"The vendor's authoritative identification of the network management
subsystem contained in the entity.   This value is allocated within the SMI
enterprises subtree (1.3.6.1.4.1) and provides an easy and unambiguous
means for determining `what kind of box' is being managed.   For example, if
vendor `Flintstones, Inc.' was assigned the subtree 1.3.6.1.4.1.4242, it could
assign the identifier 1.3.6.1.4.1.4242.1.1 to its `Fred Router'."

        ::= { system 2 }

sysUpTime OBJECT-TYPE
        SYNTAX   TimeTicks
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION
"The time (in hundredths of a second) since the network management portion
of the system was last re-initialized."

        ::= { system 3 }

sysContact OBJECT-TYPE
        SYNTAX   DisplayString (SIZE (0..255))
        ACCESS   read-write
        STATUS   mandatory
        DESCRIPTION
"The textual identification of the contact person for this managed node,
together with information on how to contact this person."

        ::= { system 4 }

sysName OBJECT-TYPE
```

SYNTAX DisplayString (SIZE (0..255))
ACCESS read-write
STATUS mandatory
DESCRIPTION
"An administratively-assigned name for this managed node. By convention, this is the node's fully-qualified domain name."

::= { system 5 }

sysLocation OBJECT-TYPE
SYNTAX DisplayString (SIZE (0..255))
ACCESS read-write
STATUS mandatory
DESCRIPTION
"The physical location of this node (e.g., `telephone closet, 3rd floor')."

::= { system 6 }

sysServices OBJECT-TYPE
SYNTAX INTEGER (0..127)
ACCESS read-only
STATUS mandatory
DESCRIPTION
"A value which indicates the set of services that this entity primarily offers.

The value is a sum. This sum initially takes the value zero, Then, for each layer, L, in the range 1 through 7, that this node performs transactions for, 2 raised to (L - 1) is added to the sum. For example, a node which performs primarily routing functions would have a value of 4 (2^(3-1)). In contrast, a node which is a host offering application services would have a value of 72 (2^(4-1) + 2^(7-1)). Note that in the context of the Internet suite of protocols, values should be calculated accordingly:

layer	functionality
1	physical (e.g., repeaters)
2	datalink/subnetwork (e.g., bridges)
3	internet (e.g., IP gateways)
4	end-to-end (e.g., IP hosts)
7	applications (e.g., mail relays)

For systems including OSI protocols, layers 5 and 6 may also be counted."

::= { system 7 }

RFC-1213 Management Information Base for Network Management - Group Definitions

The Interfaces Group

```
-- Implementation of the Interfaces group is mandatory for
-- all systems.

ifNumber OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
"The number of network interfaces (regardless of their current state) present
on this system."
        ::= { interfaces 1 }

-- the Interfaces table

-- The Interfaces table contains information on the entity's
-- interfaces.  Each interface is thought of as being
-- attached to a `subnetwork'.  Note that this term should
-- not be confused with `subnet' which refers to an
-- addressing partitioning scheme used in the Internet suite
-- of protocols.

ifTable OBJECT-TYPE
        SYNTAX  SEQUENCE OF IfEntry
        ACCESS  not-accessible
        STATUS  mandatory
        DESCRIPTION
"A list of interface entries.  The number of entries is given by the value of
ifNumber."
        ::= { interfaces 2 }

ifEntry OBJECT-TYPE
        SYNTAX  IfEntry
        ACCESS  not-accessible
        STATUS  mandatory
        DESCRIPTION
"An interface entry containing objects at the subnetwork layer and below for a
particular interface."
        INDEX   { ifIndex }
        ::= { ifTable 1 }

IfEntry ::=
        SEQUENCE {
            ifIndex
                INTEGER,
            ifDescr
                DisplayString,
            ifType
                INTEGER,
```

```
        ifMtu
            INTEGER,
        ifSpeed
            Gauge,
        ifPhysAddress
            PhysAddress,
        ifAdminStatus
            INTEGER,
        ifOperStatus
            INTEGER,
        ifLastChange
            TimeTicks,
        ifInOctets
            Counter,
        ifInUcastPkts
            Counter,
        ifInNUcastPkts
            Counter,
        ifInDiscards
            Counter,
        ifInErrors
            Counter,
        ifInUnknownProtos
            Counter,
        ifOutOctets
            Counter,
        ifOutUcastPkts
            Counter,
        ifOutNUcastPkts
            Counter,
        ifOutDiscards
            Counter,
        ifOutErrors
            Counter,
        ifOutQLen
            Gauge,
        ifSpecific
            OBJECT IDENTIFIER
    }

ifIndex OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
"A unique value for each interface.  Its value ranges between 1 and the value
of ifNumber.  The value for each interface must remain constant at least from
one re-initialization of the entity's network management system to the next
re- initialization."
    ::= { ifEntry 1 }

ifDescr OBJECT-TYPE
    SYNTAX  DisplayString (SIZE (0..255))
    ACCESS  read-only
    STATUS  mandatory
```

```
        DESCRIPTION
"A textual string containing information about the interface.   This string
should include the name of the manufacturer, the product name and the
version of the hardware interface."

        ::= { ifEntry 2 }

ifType OBJECT-TYPE
        SYNTAX   INTEGER {
                    other(1),             -- none of the following
                    regular1822(2),
                    hdh1822(3),
                    ddn-x25(4),
                    rfc877-x25(5),
                    ethernet-csmacd(6),
                    iso88023-csmacd(7),
                    iso88024-tokenBus(8),
                    iso88025-tokenRing(9),
                    iso88026-man(10),
                    starLan(11),
                    proteon-10Mbit(12),
                    proteon-80Mbit(13),
                    hyperchannel(14),
                    fddi(15),
                    lapb(16),
                    sdlc(17),
                    ds1(18),              -- T-1
                    e1(19),               -- european equiv. of T-1
                    basicISDN(20),
                    primaryISDN(21),    -- proprietary serial
                    propPointToPointSerial(22),
                    ppp(23),
                    softwareLoopback(24),
                    eon(25),                -- CLNP over IP [RFC-1070]
                    ethernet-3Mbit(26),
                    nsip(27),              -- XNS over IP
                    slip(28),             -- generic SLIP
                    ultra(29),            -- ULTRA technologies
                    ds3(30),               -- T-3
                    sip(31),              -- SMDS
                    frame-relay(32)
                }
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION
"The type of interface, distinguished according to the physical/link protocol(s)
immediately `below' the network layer in the protocol stack."

        ::= { ifEntry 3 }

ifMtu OBJECT-TYPE
        SYNTAX   INTEGER
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION
"The size of the largest datagram which can be sent/received on the interface,
```

specified in octets. For interfaces that are used for transmitting network datagrams, this is the size of the largest network datagram that can be sent on the interface."

::= { ifEntry 4 }

ifSpeed OBJECT-TYPE
SYNTAX Gauge
ACCESS read-only
STATUS mandatory
DESCRIPTION
"An estimate of the interface's current bandwidth in bits per second. For interfaces which do not vary in bandwidth or for those where no accurate estimation can be made, this object should contain the nominal bandwidth."

::= { ifEntry 5 }

ifPhysAddress OBJECT-TYPE
SYNTAX PhysAddress
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The interface's address at the protocol layer immediately `below' the network layer in the protocol stack. For interfaces which do not have such an address (e.g., a serial line), this object should contain an octet string of zero length."

::= { ifEntry 6 }

ifAdminStatus OBJECT-TYPE
SYNTAX INTEGER {
up(1), -- ready to pass packets
down(2),
testing(3) -- in some test mode
}
ACCESS read-write
STATUS mandatory
DESCRIPTION
"The desired state of the interface. The testing(3) state indicates that no operational packets can be passed."

::= { ifEntry 7 }

ifOperStatus OBJECT-TYPE
SYNTAX INTEGER {
up(1), -- ready to pass packets
down(2),
testing(3) -- in some test mode
}
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The current operational state of the interface. The testing(3) state indicates that no operational packets can be passed."

::= { ifEntry 8 }

ifLastChange OBJECT-TYPE
SYNTAX TimeTicks

ACCESS read-only
STATUS mandatory
DESCRIPTION
"The value of sysUpTime at the time the interface entered its current operational state. If the current state was entered prior to the last re-initialization of the local network management subsystem, then this object contains a zero value."

::= { ifEntry 9 }

ifInOctets OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The total number of octets received on the interface, including framing characters."

::= { ifEntry 10 }

ifInUcastPkts OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The number of subnetwork-unicast packets delivered to a higher-layer protocol."

::= { ifEntry 11 }

ifInNUcastPkts OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The number of non-unicast (i.e., subnetwork- broadcast or subnetwork-multicast) packets delivered to a higher-layer protocol."

::= { ifEntry 12 }

ifInDiscards OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The number of inbound packets which were chosen to be discarded even though no errors had been detected to prevent their being deliverable to a higher-layer protocol. One possible reason for discarding such a packet could be to free up buffer space."

::= { ifEntry 13 }

ifInErrors OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The number of inbound packets that contained errors preventing them from

being deliverable to a higher-layer protocol."

::= { ifEntry 14 }

ifInUnknownProtos OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The number of packets received via the interface which were discarded because of an unknown or unsupported protocol."

::= { ifEntry 15 }

ifOutOctets OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The total number of octets transmitted out of the interface, including framing characters."

::= { ifEntry 16 }

ifOutUcastPkts OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The total number of packets that higher-level protocols requested be transmitted to a subnetwork-unicast address, including those that were discarded or not sent."

::= { ifEntry 17 }

ifOutNUcastPkts OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The total number of packets that higher-level protocols requested be transmitted to a non- unicast (i.e., a subnetwork-broadcast or subnetwork-multicast) address, including those that were discarded or not sent."

::= { ifEntry 18 }

ifOutDiscards OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The number of outbound packets which were chosen to be discarded even though no errors had been detected to prevent their being transmitted. One possible reason for discarding such a packet could be to free up buffer space."

::= { ifEntry 19 }

ifOutErrors OBJECT-TYPE

SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The number of outbound packets that could not be transmitted because of errors."

::= { ifEntry 20 }

ifOutQLen OBJECT-TYPE
SYNTAX Gauge
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The length of the output packet queue (in packets)."

::= { ifEntry 21 }

ifSpecific OBJECT-TYPE
SYNTAX OBJECT IDENTIFIER
ACCESS read-only
STATUS mandatory
DESCRIPTION
"A reference to MIB definitions specific to the particular media being used to realize the interface. For example, if the interface is realized by an ethernet, then the value of this object refers to a document defining objects specific to ethernet. If this information is not present, its value should be set to the OBJECT IDENTIFIER { 0 0 }, which is a syntatically valid object identifier, and any conformant implementation of ASN.1 and BER must be able to generate and recognize this value."

::= { ifEntry 22 }

RFC-1213 Management Information Base for Network Management - Group Definitions

The Address Translation Group

```
-- Implementation of the Address Translation group is
-- mandatory for all systems.   Note however that this group
-- is deprecated by MIB-II. That is, it is being included
-- solely for compatibility with MIB-I nodes, and will most
-- likely be excluded from MIB-III nodes.   From MIB-II and
-- onwards, each network protocol group contains its own
-- address translation tables.

-- The Address Translation group contains one table which is
-- the union across all interfaces of the translation tables
-- for converting a NetworkAddress (e.g., an IP address) into
-- a subnetwork-specific address.   For lack of a better term,
-- this document refers to such a subnetwork-specific address
-- as a `physical' address.

-- Examples of such translation tables are: for broadcast
-- media where ARP is in use, the translation table is
-- equivalent to the ARP cache; or, on an X.25 network where
-- non-algorithmic translation to X.121 addresses is
-- required, the translation table contains the
-- NetworkAddress to X.121 address equivalences.

atTable OBJECT-TYPE
        SYNTAX   SEQUENCE OF AtEntry
        ACCESS   not-accessible
        STATUS   deprecated
        DESCRIPTION
"The Address Translation tables contain the NetworkAddress to `physical'
address equivalences. Some interfaces do not use translation tables for
determining address equivalences (e.g., DDN-X.25 has an algorithmic
method); if all interfaces are of this type, then the Address Translation table is
empty, i.e., has zero entries."
        ::= { at 1 }

atEntry OBJECT-TYPE
        SYNTAX   AtEntry
        ACCESS   not-accessible
        STATUS   deprecated
        DESCRIPTION
"Each entry contains one NetworkAddress to `physical' address equivalence."
        INDEX    { atIfIndex,
                   atNetAddress }
        ::= { atTable 1 }

AtEntry ::=
        SEQUENCE {
            atIfIndex
                 INTEGER,
            atPhysAddress
```

```
            PhysAddress,
        atNetAddress
            NetworkAddress
    }

atIfIndex OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-write
    STATUS  deprecated
    DESCRIPTION
"The interface on which this entry's equivalence is effective.  The interface
identified by a particular value of this index is the same interface as identified
by the same value of ifIndex."
    ::= { atEntry 1 }

atPhysAddress OBJECT-TYPE
    SYNTAX  PhysAddress
    ACCESS  read-write
    STATUS  deprecated
    DESCRIPTION
"The media-dependent `physical' address.

Setting this object to a null string (one of zero length) has the effect of
invaliding the corresponding entry in the atTable object.  That is, it effectively
dissasociates the interface identified with said entry from the mapping
identified with said entry.  It is an implementation-specific matter as to
whether the agent removes an invalidated entry from the table. Accordingly,
management stations must be prepared to receive tabular information from
agents that corresponds to entries not currently in use. Proper interpretation
of such entries requires examination of the relevant atPhysAddress object."
    ::= { atEntry 2 }

atNetAddress OBJECT-TYPE
    SYNTAX  NetworkAddress
    ACCESS  read-write
    STATUS  deprecated
    DESCRIPTION
"The NetworkAddress (e.g., the IP address) corresponding to the media-
dependent `physical' address."
    ::= { atEntry 3 }
```

The IP Group

```
-- Implementation of the IP group is mandatory for all
-- systems.

ipForwarding OBJECT-TYPE
        SYNTAX   INTEGER {
                    forwarding(1),      -- acting as a gateway
                    not-forwarding(2) -- NOT acting as a gateway
                 }
        ACCESS   read-write
        STATUS   mandatory
        DESCRIPTION
"The indication of whether this entity is acting as an IP gateway in respect to
the forwarding of datagrams received by, but not addressed to, this entity.   IP
gateways forward datagrams.   IP hosts do not (except those source-routed via
the host).

Note that for some managed nodes, this object may take on only a subset of
the values possible. Accordingly, it is appropriate for an agent to return a
`badValue' response if a management station attempts to change this object
to an inappropriate value."
        ::= { ip 1 }

ipDefaultTTL OBJECT-TYPE
        SYNTAX   INTEGER
        ACCESS   read-write
        STATUS   mandatory
        DESCRIPTION
"The default value inserted into the Time-To-Live field of the IP header of
datagrams originated at this entity, whenever a TTL value is not supplied by
the transport layer protocol."
        ::= { ip 2 }

ipInReceives OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION
"The total number of input datagrams received from interfaces, including
those received in error."
        ::= { ip 3 }

ipInHdrErrors OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION
"The number of input datagrams discarded due to errors in their IP headers,
including bad checksums, version number mismatch, other format errors,
time-to-live exceeded, errors discovered in processing their IP options, etc."
```

::= { ip 4 }

ipInAddrErrors OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The number of input datagrams discarded because the IP address in their IP header's destination field was not a valid address to be received at this entity. This count includes invalid addresses (e.g., 0.0.0.0) and addresses of unsupported Classes (e.g., Class E). For entities which are not IP Gateways and therefore do not forward datagrams, this counter includes datagrams discarded because the destination address was not a local address."

::= { ip 5 }

ipForwDatagrams OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The number of input datagrams for which this entity was not their final IP destination, as a result of which an attempt was made to find a route to forward them to that final destination. In entities which do not act as IP Gateways, this counter will include only those packets which were Source-Routed via this entity, and the Source- Route option processing was successful."

::= { ip 6 }

ipInUnknownProtos OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The number of locally-addressed datagrams received successfully but discarded because of an unknown or unsupported protocol."

::= { ip 7 }

ipInDiscards OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The number of input IP datagrams for which no problems were encountered to prevent their continued processing, but which were discarded (e.g., for lack of buffer space). Note that this counter does not include any datagrams discarded while awaiting re-assembly."

::= { ip 8 }

ipInDelivers OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION

"The total number of input datagrams successfully delivered to IP user-protocols (including ICMP)."

::= { ip 9 }

ipOutRequests OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The total number of IP datagrams which local IP user-protocols (including ICMP) supplied to IP in requests for transmission. Note that this counter does not include any datagrams counted in ipForwDatagrams."

::= { ip 10 }

ipOutDiscards OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The number of output IP datagrams for which no problem was encountered to prevent their transmission to their destination, but which were discarded (e.g., for lack of buffer space). Note that this counter would include datagrams counted in ipForwDatagrams if any such packets met this (discretionary) discard criterion."

::= { ip 11 }

ipOutNoRoutes OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The number of IP datagrams discarded because no route could be found to transmit them to their destination. Note that this counter includes any packets counted in ipForwDatagrams which meet this `no-route' criterion. Note that this includes any datagarms which a host cannot route because all of its default gateways are down."

::= { ip 12 }

ipReasmTimeout OBJECT-TYPE
SYNTAX INTEGER
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The maximum number of seconds which received fragments are held while they are awaiting reassembly at this entity."

::= { ip 13 }

ipReasmReqds OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The number of IP fragments received which needed to be reassembled at this

entity."

::= { ip 14 }

ipReasmOKs OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The number of IP datagrams successfully re- assembled."

::= { ip 15 }

ipReasmFails OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The number of failures detected by the IP re- assembly algorithm (for whatever reason: timed out, errors, etc). Note that this is not necessarily a count of discarded IP fragments since some algorithms (notably the algorithm in RFC 815) can lose track of the number of fragments by combining them as they are received."

::= { ip 16 }

ipFragOKs OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The number of IP datagrams that have been successfully fragmented at this entity."

::= { ip 17 }

ipFragFails OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The number of IP datagrams that have been discarded because they needed to be fragmented at this entity but could not be, e.g., because their Don't Fragment flag was set."

::= { ip 18 }

ipFragCreates OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The number of IP datagram fragments that have been generated as a result of fragmentation at this entity."

::= { ip 19 }

```
-- the IP address table

-- The IP address table contains this entity's IP addressing
-- information.

ipAddrTable OBJECT-TYPE
        SYNTAX   SEQUENCE OF IpAddrEntry
        ACCESS   not-accessible
        STATUS   mandatory
        DESCRIPTION
"The table of addressing information relevant to this entity's IP addresses."
        ::= { ip 20 }

ipAddrEntry OBJECT-TYPE
        SYNTAX   IpAddrEntry
        ACCESS   not-accessible
        STATUS   mandatory
        DESCRIPTION
"The addressing information for one of this entity's IP addresses."
        INDEX    { ipAdEntAddr }
        ::= { ipAddrTable 1 }

IpAddrEntry ::=
        SEQUENCE {
            ipAdEntAddr
                IpAddress,
            ipAdEntIfIndex
                INTEGER,
            ipAdEntNetMask
                IpAddress,
            ipAdEntBcastAddr
                INTEGER,
            ipAdEntReasmMaxSize
                INTEGER (0..65535)
        }

ipAdEntAddr OBJECT-TYPE
        SYNTAX   IpAddress
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION
"The IP address to which this entry's addressing information pertains."
        ::= { ipAddrEntry 1 }


ipAdEntIfIndex OBJECT-TYPE
        SYNTAX   INTEGER
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION
"The index value which uniquely identifies the interface to which this entry is
applicable.   The interface identified by a particular value of this index is the
same interface as identified by the same value of ifIndex."
```

```
        ::= { ipAddrEntry 2 }

ipAdEntNetMask OBJECT-TYPE
        SYNTAX   IpAddress
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION
"The subnet mask associated with the IP address of this entry.   The value of
the mask is an IP address with all the network bits set to 1 and all the hosts
bits set to 0."
        ::= { ipAddrEntry 3 }

ipAdEntBcastAddr OBJECT-TYPE
        SYNTAX   INTEGER
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION
"The value of the least-significant bit in the IP broadcast address used for
sending datagrams on the (logical) interface associated with the IP address of
this entry.   For example, when the Internet standard all-ones broadcast
address is used, the value will be 1.   This value applies to both the subnet
and network broadcasts addresses used by the entity on this (logical)
interface."
        ::= { ipAddrEntry 4 }

ipAdEntReasmMaxSize OBJECT-TYPE
        SYNTAX   INTEGER (0..65535)
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION
"The size of the largest IP datagram which this entity can re-assemble from
incoming IP fragmented datagrams received on this interface."
        ::= { ipAddrEntry 5 }

-- the IP routing table

-- The IP routing table contains an entry for each route
-- presently known to this entity.

ipRouteTable OBJECT-TYPE
        SYNTAX   SEQUENCE OF IpRouteEntry
        ACCESS   not-accessible
        STATUS   mandatory
        DESCRIPTION
"This entity's IP Routing table."
        ::= { ip 21 }

ipRouteEntry OBJECT-TYPE
        SYNTAX   IpRouteEntry
        ACCESS   not-accessible
        STATUS   mandatory
        DESCRIPTION
"A route to a particular destination."
```

```
        INDEX    { ipRouteDest }
        ::= { ipRouteTable 1 }

IpRouteEntry ::=
        SEQUENCE {
            ipRouteDest
                IpAddress,
            ipRouteIfIndex
                INTEGER,
            ipRouteMetric1
                INTEGER,
            ipRouteMetric2
                INTEGER,
            ipRouteMetric3
                INTEGER,
            ipRouteMetric4
                INTEGER,
            ipRouteNextHop
                IpAddress,
            ipRouteType
                INTEGER,
            ipRouteProto
                INTEGER,
            ipRouteAge
                INTEGER,
            ipRouteMask
                IpAddress,
            ipRouteMetric5
                INTEGER,
            ipRouteInfo
                OBJECT IDENTIFIER
        }

ipRouteDest OBJECT-TYPE
        SYNTAX   IpAddress
        ACCESS   read-write
        STATUS   mandatory
        DESCRIPTION
"The destination IP address of this route.   An entry with a value of 0.0.0.0 is
considered a default route.   Multiple routes to a single destination can appear
in the table, but access to such multiple entries is dependent on the table-
access mechanisms defined by the network management protocol in use."
        ::= { ipRouteEntry 1 }

ipRouteIfIndex OBJECT-TYPE
        SYNTAX   INTEGER
        ACCESS   read-write
        STATUS   mandatory
        DESCRIPTION
"The index value which uniquely identifies the local interface through which
the next hop of this route should be reached.   The interface identified by a
particular value of this index is the same interface as identified by the same
value of ifIndex."
        ::= { ipRouteEntry 2 }
```

```
ipRouteMetric1 OBJECT-TYPE
        SYNTAX   INTEGER
        ACCESS   read-write
        STATUS   mandatory
        DESCRIPTION
"The primary routing metric for this route.   The semantics of this metric are
determined by the routing-protocol specified in the route's ipRouteProto value.
If this metric is not used, its value should be set to -1."

        ::= { ipRouteEntry 3 }

ipRouteMetric2 OBJECT-TYPE
        SYNTAX   INTEGER
        ACCESS   read-write
        STATUS   mandatory
        DESCRIPTION
"An alternate routing metric for this route.   The semantics of this metric are
determined by the routing-protocol specified in the route's ipRouteProto value.
If this metric is not used, its value should be set to -1."

        ::= { ipRouteEntry 4 }

ipRouteMetric3 OBJECT-TYPE
        SYNTAX   INTEGER
        ACCESS   read-write
        STATUS   mandatory
        DESCRIPTION
"An alternate routing metric for this route.   The semantics of this metric are
determined by the routing-protocol specified in the route's ipRouteProto value.
If this metric is not used, its value should be set to -1."

        ::= { ipRouteEntry 5 }

ipRouteMetric4 OBJECT-TYPE
        SYNTAX   INTEGER
        ACCESS   read-write
        STATUS   mandatory
        DESCRIPTION
"An alternate routing metric for this route.   The semantics of this metric are
determined by the routing-protocol specified in the route's ipRouteProto value.
If this metric is not used, its value should be set to -1."

        ::= { ipRouteEntry 6 }

ipRouteNextHop OBJECT-TYPE
        SYNTAX   IpAddress
        ACCESS   read-write
        STATUS   mandatory
        DESCRIPTION
"The IP address of the next hop of this route. (In the case of a route bound to
an interface which is realized via a broadcast media, the value of this field is
the agent's IP address on that interface.)"

        ::= { ipRouteEntry 7 }

ipRouteType OBJECT-TYPE
        SYNTAX   INTEGER {
```

```
                other(1),          -- none of the following

                invalid(2),        -- an invalidated route
                                   -- route to directly
                direct(3),         -- connected (sub-)network

                                   -- route to a non-local
                indirect(4)        -- host/network/sub-network
            }
        ACCESS   read-write
        STATUS   mandatory
        DESCRIPTION
"The type of route.   Note that the values direct(3) and indirect(4) refer to the
notion of direct and indirect routing in the IP architecture.

Setting this object to the value invalid(2) has the effect of invalidating the
corresponding entry in the ipRouteTable object.   That is, it effectively
dissasociates the destination identified with said entry from the route
identified with said entry.   It is an implementation-specific matter as to
whether the agent removes an invalidated entry from the table. Accordingly,
management stations must be prepared to receive tabular information from
agents that corresponds to entries not currently in use. Proper interpretation
of such entries requires examination of the relevant ipRouteType object."
        ::= { ipRouteEntry 8 }

ipRouteProto OBJECT-TYPE
        SYNTAX   INTEGER {
                other(1),          -- none of the following

                                   -- non-protocol information,
                                   -- e.g., manually configured
                local(2),          -- entries

                                   -- set via a network
                netmgmt(3),        -- management protocol

                                   -- obtained via ICMP,
                icmp(4),           -- e.g., Redirect

                                   -- the remaining values are
                                   -- all gateway routing
                                   -- protocols
                egp(5),
                ggp(6),
                hello(7),
                rip(8),
                is-is(9),
                es-is(10),
                ciscoIgrp(11),
                bbnSpfIgp(12),
                ospf(13),
                bgp(14)
            }
        ACCESS   read-only
        STATUS   mandatory
```

DESCRIPTION
"The routing mechanism via which this route was learned. Inclusion of values for gateway routing protocols is not intended to imply that hosts should support those protocols."

::= { ipRouteEntry 9 }

ipRouteAge OBJECT-TYPE
SYNTAX INTEGER
ACCESS read-write
STATUS mandatory
DESCRIPTION
"The number of seconds since this route was last updated or otherwise determined to be correct. Note that no semantics of `too old' can be implied except through knowledge of the routing protocol by which the route was learned."

::= { ipRouteEntry 10 }

ipRouteMask OBJECT-TYPE
SYNTAX IpAddress
ACCESS read-write
STATUS mandatory
DESCRIPTION
"Indicate the mask to be logical-ANDed with the destination address before being compared to the value in the ipRouteDest field. For those systems that do not support arbitrary subnet masks, an agent constructs the value of the ipRouteMask by determining whether the value of the correspondent ipRouteDest field belong to a class-A, B, or C network, and then using one of:

mask	**network**
255.0.0.0	class-A
255.255.0.0	class-B
255.255.255.0	class-C

If the value of the ipRouteDest is 0.0.0.0 (a default route), then the mask value is also 0.0.0.0. It should be noted that all IP routing subsystems implicitly use this mechanism."

::= { ipRouteEntry 11 }

ipRouteMetric5 OBJECT-TYPE
SYNTAX INTEGER
ACCESS read-write
STATUS mandatory
DESCRIPTION
"An alternate routing metric for this route. The semantics of this metric are determined by the routing-protocol specified in the route's ipRouteProto value. If this metric is not used, its value should be set to -1."

::= { ipRouteEntry 12 }

ipRouteInfo OBJECT-TYPE
SYNTAX OBJECT IDENTIFIER
ACCESS read-only
STATUS mandatory
DESCRIPTION
"A reference to MIB definitions specific to the particular routing protocol which

```
is responsible for this route, as determined by the value specified in the
route's ipRouteProto value.   If this information is not present, its value should
be set to the OBJECT IDENTIFIER { 0 0 }, which is a syntatically valid object
identifier, and any conformant implementation of ASN.1 and BER must be able
to generate and recognize this value."
        ::= { ipRouteEntry 13 }


-- the IP Address Translation table

-- The IP address translation table contain the IpAddress to
-- `physical' address equivalences.   Some interfaces do not
-- use translation tables for determining address
-- equivalences (e.g., DDN-X.25 has an algorithmic method);
-- if all interfaces are of this type, then the Address
-- Translation table is empty, i.e., has zero entries.

ipNetToMediaTable OBJECT-TYPE
        SYNTAX   SEQUENCE OF IpNetToMediaEntry
        ACCESS   not-accessible
        STATUS   mandatory
        DESCRIPTION
"The IP Address Translation table used for mapping from IP addresses to
physical addresses."
        ::= { ip 22 }

ipNetToMediaEntry OBJECT-TYPE
        SYNTAX   IpNetToMediaEntry
        ACCESS   not-accessible
        STATUS   mandatory
        DESCRIPTION
"Each entry contains one IpAddress to `physical' address equivalence."
        INDEX    { ipNetToMediaIfIndex,
                   ipNetToMediaNetAddress }
        ::= { ipNetToMediaTable 1 }

IpNetToMediaEntry ::=
        SEQUENCE {
            ipNetToMediaIfIndex
                INTEGER,
            ipNetToMediaPhysAddress
                PhysAddress,
            ipNetToMediaNetAddress
                IpAddress,
            ipNetToMediaType
                INTEGER
        }

ipNetToMediaIfIndex OBJECT-TYPE
        SYNTAX   INTEGER
        ACCESS   read-write
        STATUS   mandatory
        DESCRIPTION
"The interface on which this entry's equivalence is effective.   The interface
```

identified by a particular value of this index is the same interface as identified by the same value of ifIndex."

::= { ipNetToMediaEntry 1 }

ipNetToMediaPhysAddress OBJECT-TYPE
SYNTAX PhysAddress
ACCESS read-write
STATUS mandatory
DESCRIPTION
"The media-dependent `physical' address."

::= { ipNetToMediaEntry 2 }

ipNetToMediaNetAddress OBJECT-TYPE
SYNTAX IpAddress
ACCESS read-write
STATUS mandatory
DESCRIPTION
"The IpAddress corresponding to the media- dependent `physical' address."

::= { ipNetToMediaEntry 3 }

ipNetToMediaType OBJECT-TYPE
SYNTAX INTEGER {
other(1), -- none of the following
invalid(2), -- an invalidated mapping
dynamic(3),
static(4)
}
ACCESS read-write
STATUS mandatory
DESCRIPTION
"The type of mapping.

Setting this object to the value invalid(2) has the effect of invalidating the corresponding entry in the ipNetToMediaTable. That is, it effectively dissasociates the interface identified with said entry from the mapping identified with said entry. It is an implementation-specific matter as to whether the agent removes an invalidated entry from the table. Accordingly, management stations must be prepared to receive tabular information from agents that corresponds to entries not currently in use. Proper interpretation of such entries requires examination of the relevant ipNetToMediaType object."

::= { ipNetToMediaEntry 4 }

-- additional IP objects

ipRoutingDiscards OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The number of routing entries which were chosen to be discarded even though they are valid. One possible reason for discarding such an entry could

be to free-up buffer space for other routing entries."

::= { ip 23 }

RFC-1213 Management Information Base for Network Management - Group Definitions

The ICMP Group

```
-- Implementation of the ICMP group is mandatory for all
-- systems.

icmpInMsgs OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
"The total number of ICMP messages which the entity received.  Note that
this counter includes all those counted by icmpInErrors."
        ::= { icmp 1 }

icmpInErrors OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
"The number of ICMP messages which the entity received but determined as
having ICMP-specific errors (bad ICMP checksums, bad length, etc.)."
        ::= { icmp 2 }

icmpInDestUnreachs OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
"The number of ICMP Destination Unreachable messages received."
        ::= { icmp 3 }

icmpInTimeExcds OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
"The number of ICMP Time Exceeded messages received."
        ::= { icmp 4 }


icmpInParmProbs OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
"The number of ICMP Parameter Problem messages received."
        ::= { icmp 5 }

icmpInSrcQuenchs OBJECT-TYPE
```

```
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION
"The number of ICMP Source Quench messages received."
        ::= { icmp 6 }

icmpInRedirects OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION
"The number of ICMP Redirect messages received."
        ::= { icmp 7 }

icmpInEchos OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION
"The number of ICMP Echo (request) messages received."
        ::= { icmp 8 }

icmpInEchoReps OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION
"The number of ICMP Echo Reply messages received."
        ::= { icmp 9 }

icmpInTimestamps OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION
"The number of ICMP Timestamp (request) messages received."
        ::= { icmp 10 }

icmpInTimestampReps OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION
"The number of ICMP Timestamp Reply messages received."
        ::= { icmp 11 }

icmpInAddrMasks OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION
```

"The number of ICMP Address Mask Request messages received."

::= { icmp 12 }

icmpInAddrMaskReps OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The number of ICMP Address Mask Reply messages received."

::= { icmp 13 }

icmpOutMsgs OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The total number of ICMP messages which this entity attempted to send. Note that this counter includes all those counted by icmpOutErrors."

::= { icmp 14 }

icmpOutErrors OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The number of ICMP messages which this entity did not send due to problems discovered within ICMP such as a lack of buffers. This value should not include errors discovered outside the ICMP layer such as the inability of IP to route the resultant datagram. In some implementations there may be no types of error which contribute to this counter's value."

::= { icmp 15 }

icmpOutDestUnreachs OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The number of ICMP Destination Unreachable messages sent."

::= { icmp 16 }

icmpOutTimeExcds OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The number of ICMP Time Exceeded messages sent."

::= { icmp 17 }

icmpOutParmProbs OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory

DESCRIPTION
"The number of ICMP Parameter Problem messages sent."

::= { icmp 18 }

icmpOutSrcQuenchs OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The number of ICMP Source Quench messages sent."

::= { icmp 19 }

icmpOutRedirects OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The number of ICMP Redirect messages sent. For a host, this object will always be zero, since hosts do not send redirects."

::= { icmp 20 }

icmpOutEchos OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The number of ICMP Echo (request) messages sent."

::= { icmp 21 }

icmpOutEchoReps OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The number of ICMP Echo Reply messages sent."

::= { icmp 22 }

icmpOutTimestamps OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The number of ICMP Timestamp (request) messages sent."

::= { icmp 23 }

icmpOutTimestampReps OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The number of ICMP Timestamp Reply messages sent."

::= { icmp 24 }

```
icmpOutAddrMasks OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION
"The number of ICMP Address Mask Request messages sent."
        ::= { icmp 25 }


icmpOutAddrMaskReps OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION
"The number of ICMP Address Mask Reply messages sent."
        ::= { icmp 26 }
```

The TCP Group

```
-- Implementation of the TCP group is mandatory for all
-- systems that implement the TCP.

-- Note that instances of object types that represent
-- information about a particular TCP connection are
-- transient; they persist only as long as the connection
-- in question.

tcpRtoAlgorithm OBJECT-TYPE
        SYNTAX  INTEGER {
                    other(1),      -- none of the following
                    constant(2), -- a constant rto
                    rsre(3),       -- MIL-STD-1778, Appendix B
                    vanj(4)        -- Van Jacobson's algorithm [10]
                }
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
"The algorithm used to determine the timeout value used for retransmitting
unacknowledged octets."
        ::= { tcp 1 }

tcpRtoMin OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
"The minimum value permitted by a TCP implementation for the
retransmission timeout, measured in milliseconds.  More refined semantics
for objects of this type depend upon the algorithm used to determine the
retransmission timeout.  In particular, when the timeout algorithm is rsre(3),
an object of this type has the semantics of the LBOUND quantity described in
RFC 793."
        ::= { tcp 2 }


tcpRtoMax OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
"The maximum value permitted by a TCP implementation for the
retransmission timeout, measured in milliseconds.  More refined semantics
for objects of this type depend upon the algorithm used to determine the
retransmission timeout.  In particular, when the timeout algorithm is rsre(3),
an object of this type has the semantics of the UBOUND quantity described in
RFC 793."
        ::= { tcp 3 }
```

```
tcpMaxConn OBJECT-TYPE
        SYNTAX   INTEGER
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION
"The limit on the total number of TCP connections the entity can support.   In
entities where the maximum number of connections is dynamic, this object
should contain the value -1."

        ::= { tcp 4 }

tcpActiveOpens OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION
"The number of times TCP connections have made a direct transition to the
SYN-SENT state from the CLOSED state."

        ::= { tcp 5 }

tcpPassiveOpens OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION
"The number of times TCP connections have made a direct transition to the
SYN-RCVD state from the LISTEN state."

        ::= { tcp 6 }
tcpAttemptFails OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION
"The number of times TCP connections have made a direct transition to the
CLOSED state from either the SYN-SENT state or the SYN-RCVD state, plus the
number of times TCP connections have made a direct transition to the LISTEN
state from the SYN-RCVD state."

        ::= { tcp 7 }

tcpEstabResets OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION
"The number of times TCP connections have made a direct transition to the
CLOSED state from either the ESTABLISHED state or the CLOSE-WAIT state."

        ::= { tcp 8 }

tcpCurrEstab OBJECT-TYPE
        SYNTAX   Gauge
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION
```

"The number of TCP connections for which the current state is either ESTABLISHED or CLOSE- WAIT."

::= { tcp 9 }

tcpInSegs OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The total number of segments received, including those received in error. This count includes segments received on currently established connections."

::= { tcp 10 }

tcpOutSegs OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The total number of segments sent, including those on current connections but excluding those containing only retransmitted octets."

::= { tcp 11 }

tcpRetransSegs OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The total number of segments retransmitted - that is, the number of TCP segments transmitted containing one or more previously transmitted octets."

::= { tcp 12 }

-- the TCP Connection table

-- The TCP connection table contains information about this
-- entity's existing TCP connections.

tcpConnTable OBJECT-TYPE
SYNTAX SEQUENCE OF TcpConnEntry
ACCESS not-accessible
STATUS mandatory
DESCRIPTION
"A table containing TCP connection-specific information."

::= { tcp 13 }

tcpConnEntry OBJECT-TYPE
SYNTAX TcpConnEntry
ACCESS not-accessible
STATUS mandatory
DESCRIPTION
"Information about a particular current TCP connection. An object of this type is transient, in that it ceases to exist when (or soon after) the connection makes the transition to the CLOSED state."

```
        INDEX    { tcpConnLocalAddress,
                   tcpConnLocalPort,
                   tcpConnRemAddress,
                   tcpConnRemPort }
        ::= { tcpConnTable 1 }

TcpConnEntry ::=
        SEQUENCE {
           tcpConnState
                INTEGER,
           tcpConnLocalAddress
                IpAddress,
           tcpConnLocalPort
                INTEGER (0..65535),
           tcpConnRemAddress
                IpAddress,
           tcpConnRemPort
                INTEGER (0..65535)
        }

tcpConnState OBJECT-TYPE
        SYNTAX   INTEGER {
                   closed(1),
                   listen(2),
                   synSent(3),
                   synReceived(4),
                   established(5),
                   finWait1(6),
                   finWait2(7),
                   closeWait(8),
                   lastAck(9),
                   closing(10),
                   timeWait(11),
                   deleteTCB(12)
                 }
        ACCESS   read-write
        STATUS   mandatory
        DESCRIPTION
"The state of this TCP connection.

The only value which may be set by a management station is deleteTCB(12).
Accordingly, it is appropriate for an agent to return a `badValue' response if a
management station attempts to set this object to any other value.

If a management station sets this object to the value deleteTCB(12), then this
has the effect of deleting the TCB (as defined in RFC 793) of the corresponding
connection on the managed node, resulting in immediate termination of the
connection.

As an implementation-specific option, a RST segment may be sent from the
managed node to the other TCP endpoint (note however that RST segments
are not sent reliably)."

        ::= { tcpConnEntry 1 }

tcpConnLocalAddress OBJECT-TYPE
        SYNTAX   IpAddress
```

```
        ACCESS    read-only
        STATUS    mandatory
        DESCRIPTION
"The local IP address for this TCP connection.    In the case of a connection in
the listen state which is willing to accept connections for any IP interface
associated with the node, the value 0.0.0.0 is used."

        ::= { tcpConnEntry 2 }

tcpConnLocalPort OBJECT-TYPE
        SYNTAX    INTEGER (0..65535)
        ACCESS    read-only
        STATUS    mandatory
        DESCRIPTION
"The local port number for this TCP connection."

        ::= { tcpConnEntry 3 }

tcpConnRemAddress OBJECT-TYPE
        SYNTAX    IpAddress
        ACCESS    read-only
        STATUS    mandatory
        DESCRIPTION
"The remote IP address for this TCP connection."

        ::= { tcpConnEntry 4 }

tcpConnRemPort OBJECT-TYPE
        SYNTAX    INTEGER (0..65535)
        ACCESS    read-only
        STATUS    mandatory
        DESCRIPTION
"The remote port number for this TCP connection."

        ::= { tcpConnEntry 5 }


-- additional TCP objects

tcpInErrs OBJECT-TYPE
        SYNTAX    Counter
        ACCESS    read-only
        STATUS    mandatory
        DESCRIPTION
"The total number of segments received in error (e.g., bad TCP checksums)."

        ::= { tcp 14 }

tcpOutRsts OBJECT-TYPE
        SYNTAX    Counter
        ACCESS    read-only
        STATUS    mandatory
        DESCRIPTION
"The number of TCP segments sent containing the RST flag."

        ::= { tcp 15 }
```

RFC-1213 Management Information Base for Network Management - Group Definitions

The UDP Group

```
-- Implementation of the UDP group is mandatory for all
-- systems which implement the UDP.

udpInDatagrams OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION
"The total number of UDP datagrams delivered to UDP users."
        ::= { udp 1 }

udpNoPorts OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION
"The total number of received UDP datagrams for which there was no
application at the destination port."
        ::= { udp 2 }

udpInErrors OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION
"The number of received UDP datagrams that could not be delivered for
reasons other than the lack of an application at the destination port."
        ::= { udp 3 }
udpOutDatagrams OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION
"The total number of UDP datagrams sent from this entity."
        ::= { udp 4 }


-- the UDP Listener table

-- The UDP listener table contains information about this
-- entity's UDP end-points on which a local application is
-- currently accepting datagrams.

udpTable OBJECT-TYPE
        SYNTAX   SEQUENCE OF UdpEntry
        ACCESS   not-accessible
        STATUS   mandatory
        DESCRIPTION
```

"A table containing UDP listener information."

::= { udp 5 }

udpEntry OBJECT-TYPE
SYNTAX UdpEntry
ACCESS not-accessible
STATUS mandatory
DESCRIPTION
"Information about a particular current UDP listener."
INDEX { udpLocalAddress, udpLocalPort }
::= { udpTable 1 }

UdpEntry ::=
SEQUENCE {
udpLocalAddress
IpAddress,
udpLocalPort
INTEGER (0..65535)
}

udpLocalAddress OBJECT-TYPE
SYNTAX IpAddress
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The local IP address for this UDP listener. In the case of a UDP listener which is willing to accept datagrams for any IP interface associated with the node, the value 0.0.0.0 is used."

::= { udpEntry 1 }

udpLocalPort OBJECT-TYPE
SYNTAX INTEGER (0..65535)
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The local port number for this UDP listener."

::= { udpEntry 2 }

The EGP Group

```
-- Implementation of the EGP group is mandatory for all
-- systems which implement the EGP.

egpInMsgs OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION
"The number of EGP messages received without error."
        ::= { egp 1 }

egpInErrors OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION
"The number of EGP messages received that proved to be in error."
        ::= { egp 2 }

egpOutMsgs OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION
"The total number of locally generated EGP messages."
        ::= { egp 3 }

egpOutErrors OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION
"The number of locally generated EGP messages not sent due to resource
limitations within an EGP entity."
        ::= { egp 4 }


-- the EGP Neighbor table

-- The EGP neighbor table contains information about this
-- entity's EGP neighbors.

egpNeighTable OBJECT-TYPE
        SYNTAX   SEQUENCE OF EgpNeighEntry
        ACCESS   not-accessible
        STATUS   mandatory
        DESCRIPTION
"The EGP neighbor table."
```

```
        ::= { egp 5 }

egpNeighEntry OBJECT-TYPE
        SYNTAX   EgpNeighEntry
        ACCESS   not-accessible
        STATUS   mandatory
        DESCRIPTION
"Information about this entity's relationship with a particular EGP neighbor."
        INDEX    { egpNeighAddr }
        ::= { egpNeighTable 1 }

EgpNeighEntry ::=
        SEQUENCE {
            egpNeighState
                INTEGER,
            egpNeighAddr
                IpAddress,
            egpNeighAs
                INTEGER,
            egpNeighInMsgs
                Counter,
            egpNeighInErrs
                Counter,
            egpNeighOutMsgs
                Counter,
            egpNeighOutErrs
                Counter,
            egpNeighInErrMsgs
                Counter,
            egpNeighOutErrMsgs
                Counter,
            egpNeighStateUps
                Counter,
            egpNeighStateDowns
                Counter,
            egpNeighIntervalHello
                INTEGER,
            egpNeighIntervalPoll
                INTEGER,
            egpNeighMode
                INTEGER,
            egpNeighEventTrigger
                INTEGER
        }

egpNeighState OBJECT-TYPE
        SYNTAX   INTEGER {
                    idle(1),
                    acquisition(2),
                    down(3),
                    up(4),
                    cease(5)
                 }
        ACCESS   read-only
        STATUS   mandatory
```

DESCRIPTION
"The EGP state of the local system with respect to this entry's EGP neighbor. Each EGP state is represented by a value that is one greater than the numerical value associated with said state in RFC 904."

::= { egpNeighEntry 1 }

egpNeighAddr OBJECT-TYPE
SYNTAX IpAddress
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The IP address of this entry's EGP neighbor."

::= { egpNeighEntry 2 }

egpNeighAs OBJECT-TYPE
SYNTAX INTEGER
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The autonomous system of this EGP peer. Zero should be specified if the autonomous system number of the neighbor is not yet known."

::= { egpNeighEntry 3 }

egpNeighInMsgs OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The number of EGP messages received without error from this EGP peer."

::= { egpNeighEntry 4 }

egpNeighInErrs OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The number of EGP messages received from this EGP peer that proved to be in error (e.g., bad EGP checksum)."

::= { egpNeighEntry 5 }

egpNeighOutMsgs OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The number of locally generated EGP messages to this EGP peer."

::= { egpNeighEntry 6 }

egpNeighOutErrs OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory

```
        DESCRIPTION
"The number of locally generated EGP messages not sent to this EGP peer due
to resource limitations within an EGP entity."

        ::= { egpNeighEntry 7 }

egpNeighInErrMsgs OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION
"The number of EGP-defined error messages received from this EGP peer."

        ::= { egpNeighEntry 8 }

egpNeighOutErrMsgs OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION
"The number of EGP-defined error messages sent to this EGP peer."

        ::= { egpNeighEntry 9 }

egpNeighStateUps OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION
"The number of EGP state transitions to the UP state with this EGP peer."

        ::= { egpNeighEntry 10 }

egpNeighStateDowns OBJECT-TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION
"The number of EGP state transitions from the UP state to any other state with
this EGP peer."

        ::= { egpNeighEntry 11 }

egpNeighIntervalHello OBJECT-TYPE
        SYNTAX   INTEGER
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION
"The interval between EGP Hello command retransmissions (in hundredths of
a second).   This represents the t1 timer as defined in RFC 904."

        ::= { egpNeighEntry 12 }

egpNeighIntervalPoll OBJECT-TYPE
        SYNTAX   INTEGER
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION
```

"The interval between EGP poll command retransmissions (in hundredths of a second). This represents the t3 timer as defined in RFC 904."

::= { egpNeighEntry 13 }

egpNeighMode OBJECT-TYPE
SYNTAX INTEGER { active(1), passive(2) }
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The polling mode of this EGP entity, either passive or active."

::= { egpNeighEntry 14 }

egpNeighEventTrigger OBJECT-TYPE
SYNTAX INTEGER { start(1), stop(2) }
ACCESS read-write
STATUS mandatory
DESCRIPTION
"A control variable used to trigger operator- initiated Start and Stop events. When read, this variable always returns the most recent value that egpNeighEventTrigger was set to. If it has not been set since the last initialization of the network management subsystem on the node, it returns a value of `stop'.

When set, this variable causes a Start or Stop event on the specified neighbor, as specified on pages 8-10 of RFC 904. Briefly, a Start event causes an Idle peer to begin neighbor acquisition and a non-Idle peer to reinitiate neighbor acquisition. A stop event causes a non-Idle peer to return to the Idle state until a Start event occurs, either via egpNeighEventTrigger or otherwise."

::= { egpNeighEntry 15 }

-- additional EGP objects

egpAs OBJECT-TYPE
SYNTAX INTEGER
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The autonomous system number of this EGP entity."

::= { egp 6 }

RFC-1213 Management Information Base for Network Management - Group Definitions

The Transmission Group

```
-- Based on the transmission media underlying each interface
-- on a system, the corresponding portion of the Transmission
-- group is mandatory for that system.

-- When Internet-standard definitions for managing
-- transmission media are defined, the transmission group is
-- used to provide a prefix for the names of those objects.

-- Typically, such definitions reside in the experimental
-- portion of the MIB until they are "proven", then as a
-- part of the Internet standardization process, the
-- definitions are accordingly elevated and a new object
-- identifier, under the transmission group is defined. By
-- convention, the name assigned is:
--
--       type OBJECT IDENTIFIER      ::= { transmission number }
--
-- where "type" is the symbolic value used for the media in
-- the ifType column of the ifTable object, and "number" is
-- the actual integer value corresponding to the symbol.
```

RFC-1213 Management Information Base for Network Management - Group Definitions

The SNMP Group

```
-- Implementation of the SNMP group is mandatory for all
-- systems which support an SNMP protocol entity.  Some of
-- the objects defined below will be zero-valued in those
-- SNMP implementations that are optimized to support only
-- those functions specific to either a management agent or
-- a management station.  In particular, it should be
-- observed that the objects below refer to an SNMP entity,
-- and there may be several SNMP entities residing on a
-- managed node (e.g., if the node is hosting acting as
-- a management station).

snmpInPkts OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
"The total number of Messages delivered to the SNMP entity from the
transport service."
        ::= { snmp 1 }

snmpOutPkts OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
"The total number of SNMP Messages which were passed from the SNMP
protocol entity to the transport service."
        ::= { snmp 2 }

snmpInBadVersions OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
"The total number of SNMP Messages which were delivered to the SNMP
protocol entity and were for an unsupported SNMP version."
        ::= { snmp 3 }

snmpInBadCommunityNames OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
"The total number of SNMP Messages delivered to the SNMP protocol entity
which used a SNMP community name not known to said entity."
        ::= { snmp 4 }

snmpInBadCommunityUses OBJECT-TYPE
```

SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The total number of SNMP Messages delivered to the SNMP protocol entity which represented an SNMP operation which was not allowed by the SNMP community named in the Message."

::= { snmp 5 }

snmpInASNParseErrs OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The total number of ASN.1 or BER errors encountered by the SNMP protocol entity when decoding received SNMP Messages."

::= { snmp 6 }
-- { snmp 7 } is not used

snmpInTooBigs OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The total number of SNMP PDUs which were delivered to the SNMP protocol entity and for which the value of the error-status field is `tooBig'."

::= { snmp 8 }

snmpInNoSuchNames OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The total number of SNMP PDUs which were delivered to the SNMP protocol entity and for which the value of the error-status field is `noSuchName'."

::= { snmp 9 }

snmpInBadValues OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The total number of SNMP PDUs which were delivered to the SNMP protocol entity and for which the value of the error-status field is `badValue'."

::= { snmp 10 }

snmpInReadOnlys OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The total number valid SNMP PDUs which were delivered to the SNMP protocol entity and for which the value of the error-status field is `readOnly'.

It should be noted that it is a protocol error to generate an SNMP PDU which contains the value `readOnly' in the error-status field, as such this object is provided as a means of detecting incorrect implementations of the SNMP."

::= { snmp 11 }

snmpInGenErrs OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The total number of SNMP PDUs which were delivered to the SNMP protocol entity and for which the value of the error-status field is `genErr'."

::= { snmp 12 }

snmpInTotalReqVars OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The total number of MIB objects which have been retrieved successfully by the SNMP protocol entity as the result of receiving valid SNMP Get-Request and Get-Next PDUs."

::= { snmp 13 }

snmpInTotalSetVars OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The total number of MIB objects which have been altered successfully by the SNMP protocol entity as the result of receiving valid SNMP Set-Request PDUs."

::= { snmp 14 }

snmpInGetRequests OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The total number of SNMP Get-Request PDUs which have been accepted and processed by the SNMP protocol entity."

::= { snmp 15 }

snmpInGetNexts OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The total number of SNMP Get-Next PDUs which have been accepted and processed by the SNMP protocol entity."

::= { snmp 16 }

snmpInSetRequests OBJECT-TYPE

SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The total number of SNMP Set-Request PDUs which have been accepted and processed by the SNMP protocol entity."

::= { snmp 17 }

snmpInGetResponses OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The total number of SNMP Get-Response PDUs which have been accepted and processed by the SNMP protocol entity."

::= { snmp 18 }

snmpInTraps OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The total number of SNMP Trap PDUs which have been accepted and processed by the SNMP protocol entity."

::= { snmp 19 }

snmpOutTooBigs OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The total number of SNMP PDUs which were generated by the SNMP protocol entity and for which the value of the error-status field is `tooBig.'"

::= { snmp 20 }
snmpOutNoSuchNames OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The total number of SNMP PDUs which were generated by the SNMP protocol entity and for which the value of the error-status is `noSuchName'."

::= { snmp 21 }

snmpOutBadValues OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The total number of SNMP PDUs which were generated by the SNMP protocol entity and for which the value of the error-status field is `badValue'."

::= { snmp 22 }

-- { snmp 23 } is not used

snmpOutGenErrs OBJECT-TYPE
	SYNTAX	Counter
	ACCESS	read-only
	STATUS	mandatory
	DESCRIPTION
"The total number of SNMP PDUs which were generated by the SNMP protocol entity and for which the value of the error-status field is `genErr'."

	::= { snmp 24 }

snmpOutGetRequests OBJECT-TYPE
	SYNTAX	Counter
	ACCESS	read-only
	STATUS	mandatory
	DESCRIPTION
"The total number of SNMP Get-Request PDUs which have been generated by the SNMP protocol entity."

	::= { snmp 25 }

snmpOutGetNexts OBJECT-TYPE
	SYNTAX	Counter
	ACCESS	read-only
	STATUS	mandatory
	DESCRIPTION
"The total number of SNMP Get-Next PDUs which have been generated by the SNMP protocol entity."

	::= { snmp 26 }

snmpOutSetRequests OBJECT-TYPE
	SYNTAX	Counter
	ACCESS	read-only
	STATUS	mandatory
	DESCRIPTION
"The total number of SNMP Set-Request PDUs which have been generated by the SNMP protocol entity."

	::= { snmp 27 }

snmpOutGetResponses OBJECT-TYPE
	SYNTAX	Counter
	ACCESS	read-only
	STATUS	mandatory
	DESCRIPTION
"The total number of SNMP Get-Response PDUs which have been generated by the SNMP protocol entity."

	::= { snmp 28 }

snmpOutTraps OBJECT-TYPE
	SYNTAX	Counter
	ACCESS	read-only
	STATUS	mandatory
	DESCRIPTION
"The total number of SNMP Trap PDUs which have been generated by the

SNMP protocol entity."

::= { snmp 29 }

snmpEnableAuthenTraps OBJECT-TYPE
SYNTAX INTEGER { enabled(1), disabled(2) }
ACCESS read-write
STATUS mandatory
DESCRIPTION
"Indicates whether the SNMP agent process is permitted to generate authentication-failure traps. The value of this object overrides any configuration information; as such, it provides a means whereby all authentication-failure traps may be disabled.

Note that it is strongly recommended that this object be stored in non-volatile memory so that it remains constant between re-initializations of the network management system."

::= { snmp 30 }

END

Acknowledgements

This document was produced by the SNMP Working Group:

Anne Ambler, Spider
Karl Auerbach, Sun
Fred Baker, ACC
David Bridgham, Epilogue Technology
Ken Brinkerhoff
Ron Broersma, NOSC
Brian Brown, Synoptics
Jack Brown, US Army
Theodore Brunner, Bellcore
Jeff Buffum, HP
Jeffrey Buffum, HP
John Burress, Wellfleet
Jeffrey D. Case, University of Tennessee at Knoxville
Chris Chiptasso, Spartacus
Paul Ciarfella, DEC
Bob Collet
John Cook, Chipcom
Tracy Cox, Bellcore
James R. Davin, MIT-LCS
Eric Decker, cisco
Kurt Dobbins, Cabletron
Nadya El-Afandi, Network Systems
Gary Ellis, HP
Fred Engle
Mike Erlinger
Mark S. Fedor, PSI
Richard Fox, Synoptics
Karen Frisa, CMU
Stan Froyd, ACC
Chris Gunner, DEC
Fred Harris, University of Tennessee at Knoxville
Ken Hibbard, Xylogics
Ole Jacobsen, Interop
Ken Jones
Satish Joshi, Synoptics
Frank Kastenholz, Racal-Interlan
Shimshon Kaufman, Spartacus
Ken Key, University of Tennessee at Knoxville
Jim Kinder, Fibercom
Alex Koifman, BBN
Christopher Kolb, PSI
Cheryl Krupczak, NCR
Paul Langille, DEC
Martin Lee Schoffstall, PSI
Peter Lin, Vitalink
John Lunny, TWG
Carl Malamud
Gary Malkin, FTP Software, Inc.

Randy Mayhew, University of Tennessee at Knoxville
Keith McCloghrie, Hughes LAN Systems
Donna McMaster, David Systems
Lynn Monsanto, Sun
Dave Perkins, 3COM
Jim Reinstedler, Ungerman Bass
Anil Rijsinghani, DEC
Kathy Rinehart, Arnold AFB
Kary Robertson
Marshall T. Rose, PSI (chair)
L. Michael Sabo, NCSC
Jon Saperia, DEC
Greg Satz, cisco
Martin Schoffstall, PSI
John Seligson
Steve Sherry, Xyplex
Fei Shu, NEC
Sam Sjogren, TGV
Mark Sleeper, Sparta
Lance Sprung
Mike St.Johns
Bob Stewart, Xyplex
Emil Sturniold
Kaj Tesink, Bellcore
Geoff Thompson, Synoptics
Dean Throop, Data General
Bill Townsend, Xylogics
Maurice Turcotte, Racal-Milgo
Kannan Varadhou
Sudhanshu Verma, HP
Bill Versteeg, Network Research Corporation
Warren Vik, Interactive Systems
David Waitzman, BBN
Steve Waldbusser, CMU
Dan Wintringhan
David Wood
Wengyik Yeong, PSI
Jeff Young, Cray Research

In addition, the comments of the following individuals are also acknolwedged:

Craig A. Finseth, Minnesota Supercomputer Center, Inc.
Jeffrey C. Honig, Cornell University Theory Center
Philip R. Karn, Bellcore

Authors' Addresses

Keith McCloghrie
Hughes LAN Systems
1225 Charleston Road
Mountain View, CA 94043
1225 Charleston Road
Mountain View, CA 94043

Phone: (415) 966-7934
EMail: kzm@hls.com

Marshall T. Rose
Performance Systems International
5201 Great America Parkway
Suite 3106
Santa Clara, CA 95054

Phone: +1 408 562 6222
EMail: mrose@psi.com
X.500: rose, psi, us

Information processing systems - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1), International Organization for Standardization, International Standard 8824, December 1987.

Information processing systems - Open Systems Interconnection - Specification of Basic Encoding Rules for Abstract Notation One (ASN.1), International Organization for Standardization, International Standard 8825, December 1987.

Jacobson, V., "Congestion Avoidance and Control", SIGCOMM 1988, Stanford, California.

Post Office Protocol - Version 3

M. Rose
Performance Systems International
May 1991

Status of this Memo

This memo suggests a simple method for workstations to dynamically access mail from a mailbox server. This RFC specifies an IAB standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "IAB Official Protocol Standards" for the standardization state and status of this protocol. Distribution of this memo is unlimited.

RFC-1225 Post Office Protocol - Version 3

Overview

This memo is a republication of RFC 1081 which was based on RFC 918 (since revised as RFC 937). Although similar in form to the original Post Office Protocol (POP) proposed for the Internet community, the protocol discussed in this memo is similar in spirit to the ideas investigated by the MZnet project at the University of California, Irvine.

Further, substantial work was done on examining POP in a PC-based environment. This work, which resulted in additional functionality in this protocol, was performed by the ACIS Networking Systems Group at Stanford University. The author gratefully acknowledges their interest.

RFC-1225 Post Office Protocol - Version 3

Introduction

On certain types of smaller nodes in the Internet it is often impractical to maintain a message transport system (MTS). For example, a workstation may not have sufficient resources (cycles, disk space) in order to permit a SMTP server and associated local mail delivery system to be kept resident and continuously running. Similarly, it may be expensive (or impossible) to keep a personal computer interconnected to an IP-style network for long amounts of time (the node is lacking the resource known as "connectivity").

Despite this, it is often very useful to be able to manage mail on these smaller nodes, and they often support a user agent (UA) to aid the tasks of mail handling. To solve this problem, a node which can support an MTS entity offers a maildrop service to these less endowed nodes. The Post Office Protocol - Version 3 (POP3) is intended to permit a workstation to dynamically access a maildrop on a server host in a useful fashion. Usually, this means that the POP3 is used to allow a workstation to retrieve mail that the server is holding for it.

For the remainder of this memo, the term "client host" refers to a host making use of the POP3 service, while the term "server host" refers to a host which offers the POP3 service.

A Short Digression

This memo does not specify how a client host enters mail into the transport system, although a method consistent with the philosophy of this memo is presented here:

When the user agent on a client host wishes to enter a message into the transport system, it establishes an SMTP connection to its relay host (this relay host could be, but need not be, the POP3 server host for the client host).

If this method is followed, then the client host appears to the MTS as a user agent, and should NOT be regarded as a "trusted" MTS entity in any sense whatsoever. This concept, along with the role of the POP3 as a part of a split-UA model is discussed later in this memo.

Initially, the server host starts the POP3 service by listening on TCP port 110. When a client host wishes to make use of the service, it establishes a TCP connection with the server host. When the connection is established, the POP3 server sends a greeting. The client and POP3 server then exchange commands and responses (respectively) until the connection is closed or aborted.

Commands in the POP3 consist of a keyword possibly followed by an argument. All commands are terminated by a CRLF pair.

Responses in the POP3 consist of a success indicator and a keyword possibly followed by additional information. All responses are terminated by a CRLF pair. There are currently two success indicators: positive ("+OK") and negative ("-ERR").

Responses to certain commands are multi-line. In these cases, which are clearly indicated below, after sending the first line of the response and a CRLF, any additional lines are sent, each terminated by a CRLF pair. When all lines of the response have been sent, a final line is sent, consisting of a termination octet (decimal code 046, ".") and a CRLF pair. If any line of the multi-line response begins with the termination octet, the line is "byte-stuffed" by pre-pending the termination octet to that line of the response.

Hence a multi-line response is terminated with the five octets "CRLF.CRLF". When examining a multi-line response, the client checks to see if the line begins with the termination octet. If so and if octets other than CRLF follow, the the first octet of the line (the termination octet) is stripped away. If so and if CRLF immediately follows the termination character, then the response from the POP server is ended and the line containing ".CRLF" is not considered part of the multi-line response.

A POP3 session progresses through a number of states during its lifetime. Once the TCP connection has been opened and the POP3 server has sent the greeting, the session enters the AUTHORIZATION state. In this state, the client must identify itself to the POP3 server. Once the client has successfully done this, the server acquires resources associated with the client's maildrop, and the session enters the TRANSACTION state. In this state, the client requests actions on the part of the POP3 server. When the client has finished its transactions, the session enters the UPDATE state. In this state, the POP3 server releases any resources acquired during the TRANSACTION state and says goodbye. The TCP connection is then closed.

The AUTHORIZATION State

Once the TCP connection has been opened by a POP3 client, the POP3 server issues a one line greeting. This can be any string terminated by CRLF. An example might be:

S. +OK dewey POP3 server ready (Comments to: PostMaster@UDEL.EDU)

Note that this greeting is a POP3 reply. The POP3 server should always give a positive response as the greeting.

The POP3 session is now in the AUTHORIZATION state. The client must now issue the USER command. If the POP3 server responds with a positive success indicator ("+OK"), then the client may issue either the PASS command to complete the authorization, or the QUIT command to terminate the POP3 session. If the POP3 server responds with a negative success indicator ("-ERR") to the USER command, then the client may either issue a new USER command or may issue the QUIT command.

When the client issues the PASS command, the POP3 server uses the argument pair from the USER and PASS commands to determine if the client should be given access to the appropriate maildrop. If so, the POP3 server then acquires an exclusive-access lock on the maildrop. If the lock is successfully acquired, the POP3 server parses the maildrop into individual messages (read note below), determines the last message (if any) present in the maildrop that was referenced by the RETR command, and responds with a positive success indicator. The POP3 session now enters the TRANSACTION state. If the lock can not be acquired or the client should is denied access to the appropriate maildrop or the maildrop can't be parsed for some reason, the POP3 server responds with a negative success indicator. (If a lock was acquired but the POP3 server intends to respond with a negative success indicator, the POP3 server must release the lock prior to rejecting the command.) At this point, the client may either issue a new USER command and start again, or the client may issue the QUIT command.

> **Note:**
> Minimal implementations of the POP3 need only be able to break a maildrop into its component messages; they need **not** be able to parse individual messages. More advanced implementations may wish to have this capability, for reasons discussed later.

After the POP3 server has parsed the maildrop into individual messages, it assigns a message-id to each message, and notes the size of the message in octets. The first message in the maildrop is assigned a message-id of "1", the second is assigned "2", and so on, so that the n'th message in a maildrop is assigned a message-id of "n". In POP3 commands and responses, all message-id's and message sizes are expressed in base-10 (i.e., decimal).

It sets the "highest number accessed" to be that of the last message referenced by the RETR command.

RFC-1225 Post Office Protocol - Version 3: Authorization State

Commands in Authorization State

USER
PASS
QUIT

RFC-1225 Post Office Protocol - Version 3: Authorization Commands

USER name

Arguments:

A server specific user-id (required) Restrictions: may only be given in the AUTHORIZATION state after the POP3 greeting or after an unsuccessful USER or PASS command

Possible Responses:

+OK name is welcome here
-ERR never heard of name

Examples:

C: USER mrose
S: +OK mrose is a real hoopy frood
...
C: USER frated
S: -ERR sorry, frated doesn't get his mail here

PASS string

Arguments:

A server/user-id specific password (required)

Restrictions:

may only be given in the AUTHORIZATION state after a successful USER command

Possible Responses:

+OK maildrop locked and ready
-ERR invalid password
-ERR unable to lock maildrop

Examples:

C: USER mrose
S: +OK mrose is a real hoopy frood
C: PASS secret
S: +OK mrose's maildrop has 2 messages
(320 octets)
...
C: USER mrose
S: +OK mrose is a real hoopy frood
C: PASS secret
S: -ERR unable to lock mrose's maildrop, file already locked

RFC-1225 Post Office Protocol - Version 3: Authorization Commands

QUIT

Arguments:

none

Restrictions:

none

Possible Responses:

+OK
Examples:
C: QUIT
S: +OK dewey POP3 server signing off

The TRANSACTION State

Once the client has successfully identified itself to the POP3 server and the POP3 server has locked and burst the appropriate maildrop, the POP3 session is now in the TRANSACTION state. The client may now issue any of the following POP3 commands repeatedly. After each command, the POP3 server issues a response. Eventually, the client issues the QUIT command and the POP3 session enters the UPDATE state.

RFC-1225 Post Office Protocol - Version 3: Transaction State

Commands in Transaction State

STAT
LIST
RETR
DELE
NOOP
LAST
RSET

RFC-1225 Post Office Protocol - Version 3: Transaction Commands

STAT

Arguments:

none

Restrictions:

may only be given in the TRANSACTION state.

Discussion:

The POP3 server issues a positive response with a line containing information for the maildrop. This line is called a "drop listing" for that maildrop. In order to simplify parsing, all POP3 servers are required to use a certain format for drop listings. The first octets present must indicate the number of messages in the maildrop. Following this is the size of the maildrop in octets. This memo makes no requirement on what follows the maildrop size. Minimal implementations should just end that line of the response with a CRLF pair. More advanced implementations may include other information.

> **Note:**
> This memo **strongly** discourages implementations from supplying additional information in the drop listing. Other, optional, facilities are discussed later on which permit the client to parse the messages in the maildrop.

Note that messages marked as deleted are not counted in either total.

Possible Responses:

+OK nn mm

Examples:

C: STAT
S: +OK 2 320

RFC-1225 Post Office Protocol - Version 3: Transaction Commands

LIST [msg]

Arguments:

a message-id (optionally) If a message-id is given, it may **not** refer to a message marked as deleted.

Restrictions:

may only be given in the TRANSACTION state.

Discussion:

If an argument was given and the POP3 server issues a positive response with a line containing information for that message. This line is called a "scan listing" for that message.

If no argument was given and the POP3 server issues a positive response, then the response given is multi-line. After the initial +OK, for each message in the maildrop, the POP3 server responds with a line containing information for that message. This line is called a "scan listing" for that message.

In order to simplify parsing, all POP3 servers are required to use a certain format for scan listings. The first octets present must be the message-id of the message. Following the message-id is the size of the message in octets. This memo makes no requirement on what follows the message size in the scan listing. Minimal implementations should just end that line of the response with a CRLF pair. More advanced implementations may include other information, as parsed from the message.

> **Note:**
> This memo **strongly** discourages implementations from supplying additional information in the scan listing. Other, optional, facilities are discussed later on which permit the client to parse the messages in the maildrop.

Note that messages marked as deleted are not listed.

Possible Responses:

+OK scan listing follows
-ERR no such message

Examples:

```
C:  LIST
S:  +OK 2 messages (320 octets)
S:  1 120
S:  2 200
S:  .
...
C:  LIST 2
S:  +OK 2 200
...
C:  LIST 3
S:  -ERR no such message, only 2 messages in maildrop
```

RFC-1225 Post Office Protocol - Version 3: Transaction Commands

RETR msg

Arguments:

a message-id (required) This message-id may **not** refer to a message marked as deleted.

Restrictions:

may only be given in the TRANSACTION state.

Discussion:

If the POP3 server issues a positive response, then the response given is multi-line. After the initial +OK, the POP3 server sends the message corresponding to the given message-id, being careful to byte-stuff the termination character (as with all multi-line responses).

If the number associated with this message is higher than the "highest number accessed" in the maildrop, the POP3 server updates the "highest number accessed" to the number associated with this message.

Possible Responses:

+OK message follows
-ERR no such message

Examples:

C: RETR 1
S: +OK 120 octets
S: <the POP3 server sends the entire message here>
S: .

RFC-1225 Post Office Protocol - Version 3: Transaction Commands

DELE msg

Arguments:

a message-id (required) This message-id may **not** refer to a message marked as deleted.

Restrictions:

may only be given in the TRANSACTION state.

Discussion:

The POP3 server marks the message as deleted. Any future reference to the message-id associated with the message in a POP3 command generates an error. The POP3 server does not actually delete the message until the POP3 session enters the UPDATE state.

If the number associated with this message is higher than the "highest number accessed" in the maildrop, the POP3 server updates the "highest number accessed" to the number associated with this message.

Possible Responses:

+OK message deleted
-ERR no such message

Examples:

C: DELE 1
S: +OK message 1 deleted
...
C: DELE 2
S: -ERR message 2 already deleted

NOOP

Arguments:

none

Restrictions:

may only be given in the TRANSACTION state.

Discussion:

The POP3 server does nothing, it merely replies with a positive response.

Possible Responses:

+OK

Examples:

C: NOOP
S: +OK

RFC-1225 Post Office Protocol - Version 3: Transaction Commands

LAST

Arguments:

none

Restrictions:

may only be issued in the TRANSACTION state.

Discussion:

The POP3 server issues a positive response with a line containing the highest message number which accessed. Zero is returned in case no message in the maildrop has been accessed during previous transactions. A client may thereafter infer that messages, if any, numbered greater than the response to the LAST command are messages not yet accessed by the client.

Possible Response:

+OK nn

Examples:

C: STAT
S: +OK 4 320
C: LAST
S: +OK 1
C: RETR 3
S: +OK 120 octets
S: <the POP3 server sends the entire message here>
S: .
C: LAST
S: +OK 3
C: DELE 2
S: +OK message 2 deleted
C: LAST
S: +OK 3
C: RSET
S: +OK
C: LAST
S: +OK 1

RSET

Arguments:

none

Restrictions:

may only be given in the TRANSACTION state.

Discussion:

If any messages have been marked as deleted by the POP3 server, they are unmarked. The POP3 server then replies with a positive response. In addition, the "highest number accessed" is also reset to the value determined at the beginning of the POP3 session.

Possible Responses:

+OK

Examples:

C: RSET
S: +OK maildrop has 2 messages (320 octets)

The UPDATE State

When the client issues the QUIT command from the TRANSACTION state, the POP3 session enters the UPDATE state. (Note that if the client issues the QUIT command from the AUTHORIZATION state, the POP3 session terminates but does **not** enter the UPDATE state.)

Commands in Update State

QUIT

Arguments:

none

Restrictions:

none

Discussion:

The POP3 server removes all messages marked as deleted from the maildrop. It then releases the exclusive-access lock on the maildrop and replies as to the success of these operations. The TCP connection is then closed.

Possible Responses:

+OK

Examples:

C: QUIT
S: +OK dewey POP3 server signing off (maildrop empty)
...
C: QUIT
S: +OK dewey POP3 server signing off (2 messages left)
...

RFC-1225 Post Office Protocol - Version 3

Minimal POP3 Commands

USER name	valid in the AUTHORIZATION state
PASS string	
QUIT	
STAT	valid in the TRANSACTION state
LIST [msg]	
RETR msg	
DELE msg	
NOOP	
LAST	
RSET	
QUIT	valid in the UPDATE state

Optional POP3 Commands

The POP3 commands discussed above must be supported by all minimal implementations of POP3 servers.

The optional POP3 commands described below permit a POP3 client greater freedom in message handling, while preserving a simple POP3 server implementation.

> Note:
>
> This memo **strongly** encourages implementations to support these commands in lieu of developing augmented drop and scan listings. In short, the philosophy of this memo is to put intelligence in the part of the POP3 client and not the POP3 server.

TOP
RPOP

RFC-1225 Post Office Protocol - Version 3: Optional Commands

TOP msg n

Arguments:

a message-id (required) and a number. This message-id may **not** refer to a message marked as deleted.

Restrictions:

may only be given in the TRANSACTION state.

Discussion:

If the POP3 server issues a positive response, then the response given is multi-line. After the initial +OK, the POP3 server sends the headers of the message, the blank line separating the headers from the body, and then the number of lines indicated message's body, being careful to byte-stuff the termination character (as with all multi-line responses).

Note that if the number of lines requested by the POP3 client is greater than than the number of lines in the body, then the POP3 server sends the entire message.

Possible Responses:

+OK top of message follows
-ERR no such message

Examples:

C: TOP 10
S: +OK
S: <the POP3 server sends the headers of the message, a blank line, and the first 10 lines of the body of the message>
S: .
...
C: TOP 100
S: -ERR no such message

RFC-1225 Post Office Protocol - Version 3: Optional Commands

RPOP user

Arguments:

a client specific user-id (required)

Restrictions:

may only be given in the AUTHORIZATION state after a successful USER command; in addition, may only be given if the client used a reserved (privileged) TCP port to connect to the server.

Discussion:

The RPOP command may be used instead of the PASS command to authenticate access to the maildrop. In order for this command to be successful, the POP3 client must use a reserved TCP port (port < 1024) to connect tothe server. The POP3 server uses the argument pair from the USER and RPOP commands to determine if the client should be given access to the appropriate maildrop. Unlike the PASS command however, the POP3 server considers if the remote user specified by the RPOP command who resides on the POP3 client host is allowed to access the maildrop for the user specified by the USER command (e.g., on Berkeley UNIX, the .rhosts mechanism is used). With the exception of this differing in authentication, this command is identical to the PASS command.

Note that the use of this feature has allowed much wider penetration into numerous hosts on local networks (and sometimes remote networks) by those who gain illegal access to computers by guessing passwords or otherwise breaking into the system.

Possible Responses:

+OK maildrop locked and ready
-ERR permission denied

Examples:

C: USER mrose
S: +OK mrose is a real hoopy frood
C: RPOP mrose
S: +OK mrose's maildrop has 2 messages (320 octets)

Example POP3 Session

```
S: <wait for connection on TCP port 110>
...
C: <open connection>
S:    +OK dewey POP3 server ready (Comments to: PostMaster@UDEL.EDU)
C:    USER mrose
S:    +OK mrose is a real hoopy frood
C:    PASS secret
S:    +OK mrose's maildrop has 2 messages (320 octets)
C:    STAT
S:    +OK 2 320
C:    LIST
S:    +OK 2 messages (320 octets)
S:    1 120
S:    2 200
S:    .
C:    RETR 1
S:    +OK 120 octets
S:    <the POP3 server sends message 1>
S:    .
C:    DELE 1
S:    +OK message 1 deleted
C:    RETR 2
S:    +OK 200 octets
S:    <the POP3 server sends message 2>
S:    .
C:    DELE 2
S:    +OK message 2 deleted
C:    QUIT
S:    +OK dewey POP3 server signing off (maildrop empty)
C:  <close connection>
S:  <wait for next connection>
```

Message Format

All messages transmitted during a POP3 session are assumed to conform to the standard for the format of Internet text messages [RFC822].

It is important to note that the byte count for a message on the server host may differ from the octet count assigned to that message due to local conventions for designating end-of-line. Usually, during the AUTHORIZATION state of the POP3 session, the POP3 client can calculate the size of each message in octets when it parses the maildrop into messages. For example, if the POP3 server host internally represents end-of-line as a single character, then the POP3 server simply counts each occurrence of this character in a message as two octets. Note that lines in the message which start with the termination octet need not be counted twice, since the POP3 client will remove all byte-stuffed termination characters when it receives a multi-line response.

The POP and the Split-UA Model

The underlying paradigm in which the POP3 functions is that of a split-UA model. The POP3 client host, being a remote PC based workstation, acts solely as a client to the message transport system. It does not provide delivery/authentication services to others. Hence, it is acting as a UA, on behalf of the person using the workstation. Furthermore, the workstation uses SMTP to enter mail into the MTS.

In this sense, we have two UA functions which interface to the message transport system: Posting (SMTP) and Retrieval (POP3). The entity which supports this type of environment is called a split-UA (since the user agent is split between two hosts which must interoperate to provide these functions).

> **Aside:**
> Others might term this a remote-UA instead. There are arguments supporting the use of both terms.

This memo has explicitly referenced TCP as the underlying transport agent for the POP3. This need not be the case. In the MZnet split- UA, for example, personal micro-computer systems are used which do not have IP-style networking capability. To connect to the POP3 server host, a PC establishes a terminal connection using some simple protocol (PhoneNet). A program on the PC drives the connection, first establishing a login session as a normal user. The login shell for this pseudo-user is a program which drives the other half of the terminal protocol and communicates with one of two servers. Although MZnet can support several PCs, a single pseudo-user login is present on the server host. The user-id and password for this pseudo-user login is known to all members of MZnet. Hence, the first action of the login shell, after starting the terminal protocol, is to demand a USER/PASS authorization pair from the PC. This second level of authorization is used to ascertain who is interacting with the MTS. Although the server host is deemed to support a "trusted" MTS entity, PCs in MZnet are not. Naturally, the USER/PASS authorization pair for a PC is known only to the owner of the PC (in theory, at least).

After successfully verifying the identity of the client, a modified SMTP server is started, and the PC posts mail with the server host. After the QUIT command is given to the SMTP server and it terminates, a modified POP3 server is started, and the PC retrieves mail from the server host. After the QUIT command is given to the POP3 server and it terminates, the login shell for the pseudo-user terminates the terminal protocol and logs the job out. The PC then closes the terminal connection to the server host.

The SMTP server used by MZnet is modified in the sense that it knows that it's talking to a user agent and not a "trusted" entity in the message transport system. Hence, it does performs the validation activities normally performed by an entity in the MTS when it accepts a message from a UA.

The POP3 server used by MZnet is modified in the sense that it does not require a USER/PASS combination before entering the TRANSACTION state. The reason for this (of course) is that the PC has already identified itself during the second-level authorization step described above.

> **Note:**
> Truth in advertising laws require that the author of this memo state that MZnet has not actually been fully implemented. The concepts presented and proven by the project led to the notion of the MZnet split-slot model. This notion has inspired the split-UA concept described in this memo, led to the author's interest in the POP, and heavily influenced the the description of the POP3

herein.

In fact, some UAs present in the Internet already support the notion of posting directly to an SMTP server and retrieving mail directly from a POP server, even if the POP server and client resided on the same host!

Aside:
This discussion raises an issue which this memo purposedly avoids: how does SMTP know that it's talking to a "trusted" MTS entity?

Author's Address

Marshall T. Rose
Performance Systems International
5201 Great America Parkway
Suite 3106
Santa Clara, CA 95054

Phone: +1 408 562 6222

EMail: mrose@psi.com
X.500: rose, psi, us

RFC-1250 IAB Official Protocol Standards

Internet Activities Board
Jon Postel, Editor
August 1991

Status of this Memo

This memo describes the state of standardization of protocols used in the Internet as determined by the Internet Activities Board (IAB). Distribution of this memo is unlimited.

Introduction

Discussion of the standardization process and the RFC document series is presented first, followed by an explanation of the terms. The Protocols contains the lists of protocols in each stage of standardization. Finally come pointers to references and contacts for further information.

This memo is intended to be issued quarterly; please be sure the copy you are reading is current. Current copies may be obtained from the Network Information Center or from the Internet Assigned Numbers Authority (see the contact information at the end of this memo). Do not use this edition after 30-Nov-91.

See recent changes. In the official lists an asterisk (*) next to a protocol denotes that it is new to this document or has been moved from one protocol level to another.

RFC-1250 IAB Official Protocol Standards - August 1991

The Standardization Process

The Internet Activities Board maintains this list of documents that define standards for the Internet protocol suite (see RFC-1160 for an explanation of the role and organization of the IAB and its subsidiary groups, the Internet Engineering Task Force (IETF) and the Internet Research Task Force (IRTF)). The IAB provides these standards with the goal of co-ordinating the evolution of the Internet protocols; this co-ordination has become quite important as the Internet protocols are increasingly in general commercial use.

The majority of Internet protocol development and standardization activity takes place in the working groups of the Internet Engineering Task Force.

Protocols which are to become standards in the Internet go through a series of states (proposed standard, draft standard, and standard) involving increasing amounts of scrutiny and experimental testing. At each step, the Internet Engineering Steering Group (IESG) of the IETF must make a recommendation for advancement of the protocol and the IAB must ratify it. If a recommendation is not ratified, the protocol is remanded to the IETF for further work.

To allow time for the Internet community to consider and react to standardization proposals, the IAB imposes a minimum delay of 4 months before a proposed standard can be advanced to a draft standard and 6 months before a draft standard can be promoted to standard.

It is general IAB practice that no proposed standard can be promoted to draft standard without at least two independent implementations (and the recommendation of the IESG). Promotion from draft standard to standard generally requires operational experience and demonstrated interoperability of two or more implementations (and the recommendation of the IESG).

In cases where there is uncertainty as to the proper decision concerning a protocol the IAB may convene a special review committee consisting of experts from the IETF, IRTF and the IAB with the purpose of recommending an explicit action to the IAB.

Advancement of a protocol to proposed standard is an important step since it marks a protocol as a candidate for eventual standardization (it puts the protocol "on the standards track"). Advancement to draft standard is a major step which warns the community that, unless major objections are raised or flaws are discovered, the protocol is likely to be advanced to standard in six months.

Some protocols have been superseded by better ones or are otherwise unused. Such protocols are still documented in this memorandum with the designation "historic".

Because the IAB believes it is useful to document the results of early protocol research and development work, some of the RFCs document protocols which are still in an experimental condition. The protocols are designated "experimental" in this memorandum. They appear in this report as a convenience to the community and not as evidence of their standardization. Other protocols, such as those developed by other standards organizations, or by particular vendors, may be of interest or may be recommended for use in the Internet. The specifications of such protocols may be published as RFCs for the convenience of the Internet community. These protocols are labeled "informational" in this memorandum.

In addition to the working groups of the IETF, protocol development and experimentation may take place as a result of the work of the research groups of the Internet Research Task Force, or the work of other individuals interested in Internet protocol development. The IAB encourages the documentation of such experimental work in the RFC series, but none of this work is considered to be on the track for standardization until the IESG has made a

recommendation to advance the protocol to the proposed standard state, and the IAB has approved this step.

A few protocols have achieved widespread implementation without the approval of the IESG and the IAB. For example, some vendor protocols have become very important to the Internet community even though they have not been recommended by the IESG or ratified by the IAB. However, the IAB strongly recommends that the IAB standards process be used in the evolution of the protocol suite to maximize interoperability (and to prevent incompatible protocol requirements from arising). The IAB reserves the use of the terms "standard", "draft standard", and "proposed standard" in any RFC or other publication of Internet protocols to only those protocols which the IAB has approved.

In addition to a state (like "Proposed Standard"), a protocol is also assigned a status, or requirement level, in this document. The possible requirement levels ("Required", "Recommended", "Elective", "Limited Use", and "Not Recommended") are defined in Section 4.2. When a protocol is on the standards track, that is in the proposed standard, draft standard, or standard state (see Section 5), the status shown in Section 6 is the current status. For a proposed or draft standard, however, the IAB will also endeavor to indicate the eventual status this protocol will have after adoption as a standard.

Few protocols are required to be implemented in all systems; this is because there is such a variety of possible systems, for example, gateways, terminal servers, workstations, and multi-user hosts. The requirement level shown in this document is only a one word label, which may not be sufficient to characterize the implementation requirements for a protocol in all situations. For some protocols, this document contains an additional status paragraph (an applicability statement). In addition, more detailed status information is contained in separate requirements documents (see Section 3).

The Request for Comments Documents

The documents called Request for Comments (or RFCs) are the working notes of the "Network Working Group", that is the Internet research and development community. A document in this series may be on essentially any topic related to computer communication, and may be anything from a meeting report to the specification of a standard.

Notice: All standards are published as RFCs, but not all RFCs specify standards.

Anyone can submit a document for publication as an RFC. Submissions must be made via electronic mail to the RFC Editor (see the contact information at the end of this memo).

While RFCs are not refereed publications, they do receive technical review from the task forces, individual technical experts, or the RFC Editor, as appropriate.

The RFC series comprises a wide range of documents, ranging from informational documents of general interests to specifications of standard Internet protocols. In cases where submission is intended to document a proposed standard, draft standard, or standard protocol, the RFC Editor will publish the document only with the approval of both the IESG and the IAB. For documents describing experimental work, the RFC Editor will notify the IESG before publication, allowing for the possibility of review by the relevant IETF working group or IRTF research group and provide those comments to the author. See Section 5.1 for more detail.

Once a document is assigned an RFC number and published, that RFC is never revised or re-issued with the same number. There is never a question of having the most recent version of a particular RFC. However, a protocol (such as File Transfer Protocol (FTP)) may be improved and re-documented many times in several different RFCs. It is important to verify that you have the most recent RFC on a particular protocol. This "IAB Official Protocol Standards" memo is the reference for determining the correct RFC for the current specification of each protocol.

The RFCs are available from the Network Information Center at SRI International, and a number of other sites. For more information about obtaining RFCs, see Sections 7.4 and 7.5.

Other Reference Documents

There are four other reference documents of interest in checking the current status of protocol specifications and standardization. These are the Assigned Numbers, the Annotated Internet Protocols, the Gateway Requirements, and the Host Requirements. Note that these documents are revised and updated at different times; in case of differences between these documents, the most recent must prevail.

Also, one should be aware of the MIL-STD publications on IP, TCP, Telnet, FTP, and SMTP.

Assigned Numbers
Annotated Internet Protocols
Gateway Requirements
Host Requirements
The MIL-STD Documents

RFC-1250 IAB Official Protocol Standards - August 1991

Assigned Numbers

This document lists the assigned values of the parameters used in the various protocols. For example, IP protocol codes, TCP port numbers, Telnet Option Codes, ARP hardware types, and Terminal Type names. Assigned Numbers was most recently issued as RFC-1060.

Another document, Internet Numbers, lists the assigned IP network numbers, and the autonomous system numbers. Internet Numbers was most recently issued as RFC-1166.

RFC-1250 IAB Official Protocol Standards - August 1991

Annotated Internet Protocols

This document lists the protocols and describes any known problems and ongoing experiments. This document was most recently issued as RFC-1011 under the title "Official Internet Protocols".

Gateway Requirements

This document reviews the specifications that apply to gateways and supplies guidance and clarification for any ambiguities. Gateway Requirements is RFC-1009. A working group of the IETF is actively preparing a revision.

Host Requirements

This pair of documents reviews the specifications that apply to hosts and supplies guidance and clarification for any ambiguities. Host Requirements was recently issued as RFC-1122 and RFC-1123.

Note: The additions and clarifications included in this pair of RFCs have been integrated into the orginal RFCs within this system; Therefore these two RFCs are not explicitly included.

The MIL-STD Documents

The Internet community specifications for IP (RFC-791) and TCP (RFC- 793) and the DoD MIL-STD specifications are intended to describe exactly the same protocols. Any difference in the protocols specified by these sets of documents should be reported to DCA and to the IAB. The RFCs and the MIL-STDs for IP and TCP differ in style and level of detail. It is strongly advised that the two sets of documents be used together.

The IAB and the DoD MIL-STD specifications for the FTP, SMTP, and Telnet protocols are essentially the same documents (RFCs 765, 821, 854). The MIL-STD versions have been edited slightly. Note that the current Internet specification for FTP is RFC-959.

Internet Protocol (IP)	MIL-STD-1777
Transmission Control Protocol (TCP)	MIL-STD-1778
File Transfer Protocol (FTP)	MIL-STD-1780
Simple Mail Transfer Protocol (SMTP)	MIL-STD-1781
Telnet Protocol and Options (TELNET)	MIL-STD-1782

These documents are available from the Naval Publications and Forms Center. Requests can be initiated by telephone, telegraph, or mail; however, it is preferred that private industry use form DD1425, if possible. These five documents are included in the 1985 DDN Protocol Handbook (available from the Network Information Center, see Section 7.4).

Naval Publications and Forms Center, Code 3015
5801 Tabor Ave
Philadelphia, PA 19120
Phone: 1-215-697-3321 (order tape)
1-215-697-4834 (conversation)

RFC-1250 IAB Official Protocol Standards August 1991

Explanation of Terms

There are two independent categorization of protocols. The first is the STATE of standardization, one of "standard", "draft standard", "proposed standard", "experimental", "informational" or "historic". The second is the STATUS of this protocol, one of "required", "recommended", "elective", "limited use", or "not recommended".

The status or requirement level is difficult to portray in a one word label. These status labels should be considered only as an indication, and a further description, or applicability statement, should be consulted.

When a protocol is advanced to proposed standard or draft standard, it is labeled with a current status and when possible, the IAB also notes the status that the protocol is expected to have when it reaches the standard state.

At any given time a protocol occupies a cell of the following matrix. Protocols are likely to be in cells in about the following proportions (indicated by the relative number of Xs). A new protocol is most likely to start in the (proposed standard, elective) cell, or the (experimental, not recommended) cell.

State \ Status	Req	Rec	Ele	Lim	Not
Std	X	XXX	XXX		
Draft	X	X	XXX		
Prop		X	XXX	X	
Info		X	XXX	X	X
Expr			X	XXX	X
Hist				X	XXX

What is a "system"?

> Some protocols are particular to hosts and some to gateways; a few protocols are used in both. The definitions of the terms below will refer to a "system" which is either a host or a gateway (or both). It should be clear from the context of the particular protocol which types of systems are intended.

Definitions of Protocol State

Every protocol listed in this document is assigned to a STATE of standardization: "standard", "draft standard", "proposed standard", "experimental", or "historic".

Standard Protocol

The IAB has established this as an official standard protocol for the Internet. These are separated into two groups: (1) IP protocol and above, protocols that apply to the whole Internet; and (2) network-specific protocols, generally specifications of how to do IP on particular types of networks. .2. Draft Standard Protocol

The IAB is actively considering this protocol as a possible Standard Protocol. Substantial and widespread testing and comment are desired. Comments and test results should be submitted to the IAB. There is a possibility that changes will be made in a Draft Standard Protocol before it becomes a Standard Protocol.

Proposed Standard Protocol

These are protocol proposals that may be considered by the IAB for standardization in the future. Implementation and testing by several groups is desirable. Revision of the protocol specification is likely.

Experimental Protocol

A system should not implement an experimental protocol unless it is participating in the experiment and has coordinated its use of the protocol with the developer of the protocol.

Typically, experimental protocols are those that are developed as part of an ongoing research project not related to an operational service offering. While they may be proposed as a service protocol at a later stage, and thus become proposed standard, draft standard, and then standard protocols, the designation of a protocol as experimental may sometimes be meant to suggest that the protocol, although perhaps mature, is not intended for operational use.

Informational Protocol

Protocols developed by other standard organizations, or vendors, or that are for other reasons outside the purview of the IAB, may be published as RFCs for the convenience of the Internet community as informational protocols. Such protocols may in some cases also be recommended for use in the Internet by the IAB.

Historic Protocol

These are protocols that are unlikely to ever become standards in the Internet either because they have been superseded by later developments or due to lack of interest.

Definitions of Protocol Status

This document lists a STATUS for each protocol. The status is one of "required", "recommended", "elective", "limited use", or "not recommended".

Required Protocol

A system must implement the required protocols.

Recommended Protocol

A system should implement the recommended protocols.

Elective Protocol

A system may or may not implement an elective protocol. The general notion is that if you are going to do something like this, you must do exactly this. There may be several elective protocols in a general area, for example, there are several electronic mail protocols, and several routing protocols.

Limited Use Protocol

These protocols are for use in limited circumstances. This may be because of their experimental state, specialized nature, limited functionality, or historic state.

Not Recommended Protocol

These protocols are not recommended for general use. This may be because of their limited functionality, specialized nature, or experimental or historic state.

The RFC Processing Decision Table

Here is the current decision table for processing submissions by the RFC Editor. The processing depends on who submitted it, and the status they want it to have.

Desired Status	IAB	IESG	IRSG or RG	Other
	S O U R C E			
Full or Draft Standard	Publish (1)	Vote (3)	Bogus (2)	Bogus (2)
Proposed Standard	Publish (1)	Vote (3)	Refer (4)	Refer (4)
Experimental Protocol	Publish (1)	Notify (5)	Notify (5)	Notify (5)
Information or Opinion Paper	Publish (1)	Discretion (6)	Discretion (6)	Discretion (6)

(1) Publish.

(2) Bogus. Inform the source of the rules. RFCs specifying Standard, or Draft Standard must come from the IAB, only.

(3) Vote by the IAB. If approved then do Publish (1), else do Refer (4).

(4) Refer to an Area Director for review by a WG. Expect to see the document again only after approval by the IESG and the IAB.

(5) Notify both the IESG and IRSG. If no protest in 1 week then do Discretion (6), else do Refer (4).

(6) RFC Editor's discretion. The RFC Editor decides if a review is needed and if so by whom. RFC Editor decides to publish or not.

Of course, in all cases the RFC Editor can request or make minor changes for style, format,

and presentation purposes.

The IESG has designated the IESG Secretary as its agent for forwarding documents with IESG approval and for registering protest in response to notifications (5) to the RFC Editor. Documents from Area Directors or Working Group Chairs may be considered in the same way as documents from "other".

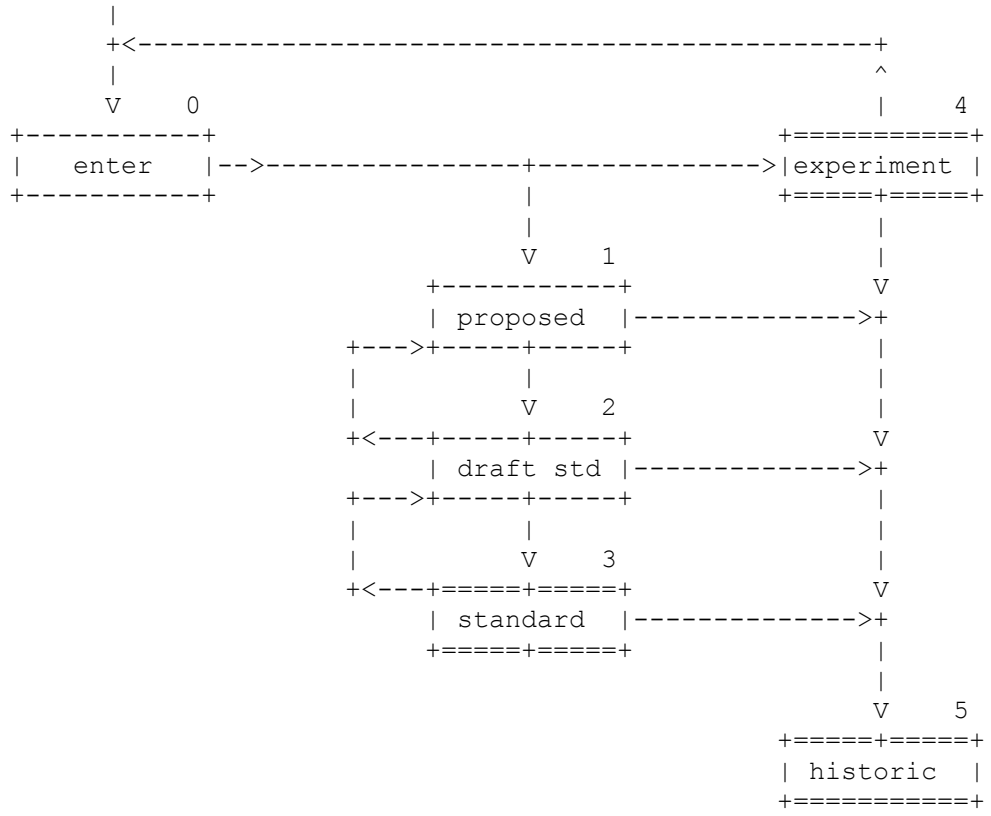
The Standards Track Diagram

There is a part of the STATUS and STATE categorization that is called the standards track. Actually, only the changes of state are significant to the progression along the standards track, though the status assignments may be changed as well.

The states illustrated by single line boxes are temporary states, those illustrated by double line boxes are long term states. A protocol will normally be expected to remain in a temporary state for several months (minimum four months for proposed standard, minimum six months for draft standard). A protocol may be in a long term state for many years.

A protocol may enter the standards track only on the recommendation of the IESG and by action of the IAB; and may move from one state to another along the track only on the recommendation of the IESG and by action of the IAB. That is, it takes both the IESG and the IAB to either start a protocol on the track or to move it along.

Generally, as the protocol enters the standards track a decision is made as to the eventual STATUS (elective, recommended, or required) the protocol will have, although a somewhat less stringent current status may be assigned, and it then is placed in the the proposed standard STATE with that status. So the initial placement of a protocol is into state 1. At any time the STATUS decision may be revisited.



The transition from proposed standard (1) to draft standard (2) can only be by action of the IAB on the recommendation of the IESG and only after the protocol has been proposed standard (1) for at least four months.

The transition from draft standard (2) to standard (3) can only be by action of the IAB on the

recommendation of the IESG and only after the protocol has been draft standard (2) for at least six months.

Occasionally, the decision may be that the protocol is not ready for standardization and will be assigned to the experimental state (4). This is off the standards track, and the protocol may be resubmitted to enter the standards track after further work. There are other paths into the experimental and historic states that do not involve IAB action.

Sometimes one protocol is replaced by another and thus becomes historic, it may happen that a protocol on the standards track is in a sense overtaken by another protocol (or other events) and becomes historic (state 5).

RFC-1250 IAB Official Protocol Standards August 1991

The Protocols

RFC-1250 IAB Official Protocol Standards August 1991; The Protocols

New RFCs

1252 OSPF Version 2 MIB

A Proposed Standard protocol.

1251 Who's Who in the Internet

This is an information document and does not specify any level of standard.

1250 This memo.

1249 DIXIE Protocol Specification

This is an information document and does not specify any level of standard.

1248 OSPF Version 2 MIB

A Proposed Standard protocol.

1247 OSPF Version 2

A Draft Standard protocol.

1246 Experience with the OSPF Protocol

This is an information document and does not specify any level of standard.

1245 OSPF Protocol Analysis

This is an information document and does not specify any level of standard.

1244 Site Security Handbook

This is an information document and does not specify any level of standard.

1243 AppleTalk Management Information Base

A Proposed Standard protocol.

1242 Benchmarking Terminology for Network Interconnection Devices

This is an information document and does not specify any level of standard.

1241 A Scheme for an Internet Encapsulation Protocol: Version 1

This is a new Experimental protocol.

1240 OSI Connectionless Transport Services on top of UDP - Version: 1

A Proposed Standard protocol.

1239 Reassignment of Experimental MIBs to Standard MIBs

A Proposed Standard protocol.

1238 CLNS MIB - for use with Connectionless Network Protocol (ISO 8473) and End System to Intermediate System (ISO 9542)

This is a new Experimental protocol.

1237 Guidelines for OSI NSAP Allocation in the Internet

A Proposed Standard protocol.

1236 IP to X.121 Address Mapping for DDN

This is an information document and does not specify any level of standard.

1235 The Coherent File Distribution Protocol
This is a new Experimental protocol.

1234 Tunneling IPX Traffic through IP Networks
A Proposed Standard protocol.

1233 Definitions of Managed Objects for the DS3 Interface Type
A Proposed Standard protocol.

1232 Definitions of Managed Objects for the DS1 Interface Type
A Proposed Standard protocol.

1231 IEEE 802.5 Token Ring MIB
A Proposed Standard protocol.

1230 IEEE 802.4 Token Bus MIB
A Proposed Standard protocol.

1229 Extensions to the Generic-Interface MIB
A Proposed Standard protocol.

1228 SNMP-DPI-Simple Network Management Protocol Distributed Program Interface
This is a new Experimental protocol.

1227 SNMP MUX Protocol and MIB
This is a new Experimental protocol.

1226 Internet Protocol Encapsulation of AX.25 Frames
This is a new Experimental protocol.

1225 Post Office Protocol - Version 3
A Draft Standard protocol.

1224 Techniques for Managing Asynchronously Generated Alerts
This is a new Experimental protocol.

1223 OSI CLNS and LLC1 Protocols on Network Systems HYPERchannel
This is an information document and does not specify any level of standard.

1222 Advancing the NSFNET Routing Architecture
This is an information document and does not specify any level of standard.

1221 Host Access Protocol (HAP) Specification - Version 2
This is an information document and does not specify any level of standard.

1220 Point-to-Point Protocol Extensions for Bridging
A Proposed Standard protocol.

1219 On the Assignment of Subnet Numbers
This is an information document and does not specify any level of standard.

Other Changes

The following are changes to protocols listed in the previous edition.

1213 Management Information Base for Network Management of TCP/IP-based internets: MIB-II

Advanced to Standard protocol.

1212 Concise MIB Definitions

Advanced to Draft Standard protocol.

Telnet Options has been added.

RFC-1250 IAB Official Protocol Standards August 1991; The Protocols

Standard Protocols

Protocol	Name	Status	RFC
--------	Assigned Numbers	Required	**1060**
--------	Gateway Requirements	Required	**1009**
--------	Host Requirements-Communications	Required	**1122**
--------	Host Requirements-Applications	Required	**1123**
IP	Internet Protocol	Required	**791**
	as amended by:		
--------	IP Subnet Extension	Required	**950**
--------	IP Broadcast Datagrams	Required	**919**
--------	IP Broadcast Datagrams with Subnets	Required	**922**
ICMP	Internet Control Message Protocol	Required	**792**
IGMP	Host Extensions for IP Multicasting	Recommended	**1112**
UDP	User Datagram Protocol	Recommended	**768**
TCP	Transmission Control Protocol	Recommended	**793**
SMI	Structure of Management Information	Recommended	**1155**
MIB	Management Information Base	Recommended	**1156**
SNMP	Simple Network Management Protocol	Recommended	**1157**
DOMAIN	Domain Name System	Recommended	**1034**, **1035**
TELNET	Telnet Protocol	Recommended	**854**
FTP	File Transfer Protocol	Recommended	**959**
SMTP	Simple Mail Transfer Protocol	Recommended	**821**
MAIL	Format of Electronic Mail Messages	Recommended	**822**
CONTENT	Content Type Header Field	Recommended	**1049**
EGP	Exterior Gateway Protocol	Recommended	**904**
ECHO	Echo Protocol	Recommended	**862**
NTP	Network Time Protocol	Recommended	**1119**
NETBIOS	NetBIOS Service Protocols	Elective	**1001**,**1002**
DISCARD	Discard Protocol	Elective	**863**
CHARGEN	Character Generator Protocol	Elective	**864**
QUOTE	Quote of the Day Protocol	Elective	**865**
USERS	Active Users Protocol	Elective	**866**
DAYTIME	Daytime Protocol	Elective	**867**
TIME	Time Server Protocol	Elective	**868**

Applicability Statements:

IGMP -- The Internet Activities Board intends to move towards general adoption of IP multicasting, as a more efficient solution than broadcasting for many applications. The host interface has been standardized in RFC-1112; however, multicast-routing gateways are in the experimental stage and are not widely available. An Internet host should support all of RFC-1112, except for the IGMP protocol itself which is optional. Even without IGMP, implementation of RFC-1112 will provide an important advance: IP-layer access to local network multicast addressing. It is expected that IGMP will become recommended for all hosts and gateways at some future date.

SMI, MIB, SNMP -- The Internet Activities Board recommends that all IP and TCP implementations be network manageable. This implies implementation of the Internet MIB (RFC-1156) and at least the recommended management protocol SNMP (RFC-1157).

RFC-1122 and **RFC-1123** have been integrated into this system by updating the **original** RFCs to which they referred, rather than by including either of them explicitly.

RFC-1250 IAB Official Protocol Standards August 1991; The Protocols

Network-Specific Standard Protocols

Protocol	Name	Status	RFC
ARP	Address Resolution Protocol	Elective	**826**
RARP	A Reverse Address Resolution Protocol	Elective	**903**
IP-ARPA	Internet Protocol on ARPANET	Elective	BBN 1822
IP-WB	Internet Protocol on Wideband Network	Elective	**907**
IP-X25	Internet Protocol on X.25 Networks	Elective	**877**
IP-E	Internet Protocol on Ethernet Networks	Elective	**894**
IP-EE	Internet Protocol on Exp. Ethernet Nets	Elective	**895**
IP-IEEE	Internet Protocol on IEEE 802	Elective	**1042**
IP-DC	Internet Protocol on DC Networks	Elective	**891**
IP-HC	Internet Protocol on Hyperchannel	Elective	**1044**
IP-ARC	Internet Protocol on ARCNET	Elective	**1051**
IP-SLIP	Transmission of IP over Serial Lines	Elective	**1055**
IP-NETBIOS	Transmission of IP over NETBIOS	Elective	**1088**
IP-FDDI	Transmission of IP over FDDI	Elective	**1103**
IP-IPX	Transmission of 802.2 over IPX Networks	Elective	**1132**

Applicability Statements:

It is expected that a system will support one or more physical networks and for each physical network supported the appropriate protocols from the above list must be supported. That is, it is elective to support any particular type of physical network, and for the physical networks actually supported it is required that they be supported exactly according to the protocols in the above list. See also the Host and Gateway Requirements RFCs for more specific information on network-specific ("link layer") protocols.

RFC-1250 IAB Official Protocol Standards August 1991; The Protocols

Draft Standard Protocols

Protocol	Name	Status	RFC	
OSPF2	Open Shortest Path First Routing V2	Elective	**1247***	
POP3	Post Office Protocol, Version 3	Elective	**1225***	
Concise-MIB	Concise MIB Definitions	Elective	**1212***	
FINGER	Finger Protocol	Elective	**1196**	
IP-FDDI	Internet Protocol on FDDI Networks	Elective	**1188**	
TOPT-LINE	Telnet Linemode Option	Elective	**1184**	
PPP	Point to Point Protocol	Elective	**1171**	
--------	Mail Privacy: Procedures	Elective	**1113**	
--------	Mail Privacy: Key Management	Elective	**1114**	
--------	Mail Privacy: Algorithms	Elective	**1115**	
BOOTP	Bootstrap Protocol	Recom.	**951**	
	Bootstrap Protocol Vendor Extendions		Recom.	**1084**
RIP	Routing Information Protocol	Elective	**1058**	
TP-TCP	ISO Transport Service on top of the TCP		Elective	**1006**
NICNAME	WhoIs Protocol	Elective	**954**	
TFTP	Trivial File Transfer Protocol	Elective	**783**	

Applicability Statements:

RIP -- The Routing Information Protocol (RIP) is widely implemented and used in the Internet. However, both implementors and users should be aware that RIP has some serious technical limitations as a routing protocol. The IETF is currently developing several candidates for a new standard "open" routing protocol with better properties than RIP. The IAB urges the Internet community to track these developments, and to implement the new protocol when it is standardized; improved Internet service will result for many users.

TP-TCP -- As OSI protocols become more widely implemented and used, there will be an increasing need to support interoperation with the TCP/IP protocols. The Internet Engineering Task Force is formulating strategies for interoperation. RFC-1006 provides one interoperation mode, in which TCP/IP is used to emulate TP0 in order to support OSI applications. Hosts that wish to run OSI connection-oriented applications in this mode should use the procedure described in RFC- 1006. In the future, the IAB expects that a major portion of the Internet will support both TCP/IP and OSI (inter-)network protocols in parallel, and it will then be possible to run OSI applications across the Internet using full OSI protocol "stacks".

PPP -- Point to Point Protocol is a method of sending IP over serial lines, which are a type of physical network. It is anticipated that PPP will be advanced to the network-specific standard protocol state in the future.

RFC-1250 IAB Official Protocol Standards August 1991; The Protocols

Proposed Standard Protocols

Protocol	Name	Status	RFC
OSPF-MIB	OSPF Version 2 MIB	Elective	**1248**,**1252***
AT- MIB	Appletalk MIB	Elective	**1243***
OSI-UDP	OSI TS on UDP	Elective	**1240***
STD-MIBs	Reassignment of Exp MIBs to Std MIBs	Elective	**1239***
OSI-NSAP	Guidelines for OSI NSAP Allocation	Elective	**1237***
IPX-IP	Tunneling IPX Traffic through IP Nets	Elective	**1234***
DS3-MIB	DS3 Interface Objects	Elective	**1233***
DS1-MIB	DS1 Interface Objects	Elective	**1232***
802.5-MIB	IEEE 802.5 Token Ring MIB	Elective	**1231***
802.4-MIP	IEEE 802.4 Token Bus MIB	Elective	**1230***
GINT-MIB	Extensions to the Generic-Interface MIB	Elective	**1229***
PPP-EXT	PPP Extensions for Bridging	Elective	**1220***
OIM-MIB-II	OSI Internet Management: MIB-II	Elective	**1214**
IP-SMDS	IP Datagrams over the SMDS Service	Elective	**1209**
IP-ARCNET	Transmitting IP Traffic over ARCNET Nets	Elective	**1201**
IS-IS	OSI IS-IS for TCP/IP Dual Environments	Elective	**1195**
IP-MTU	Path MTU Discovery	Elective	**1191**
CMOT	Common Management Information Services and Protocol over TCP/IP	Elective	**1189**
PPP-INIT	PPP Initial Configuration Options	Elective	**1172**
BGP	Border Gateway Protocol	Elective	**1163**,**1164**
IP-CMPRS	Compressing TCP/IP Headers	Elective	**1144**
ISO-TS-ECHO	Echo for ISO-8473	Elective	**1139**
SUN-NFS	Network File System Protocol	Elective	**1094**
SUN-RPC	Remote Procedure Call Protocol	Elective	**1057**
PCMAIL	Pcmail Transport Protocol	Elective	**1056**
NFILE	A File Access Protocol	Elective	**1037**
-------	Mapping between X.400(84) and RFC-822	Elective	**987**,**1026**
NNTP	Network News Transfer Protocol	Elective	**977**
HOSTNAME	HOSTNAME Protocol	Elective	**953**
SFTP	Simple File Transfer Protocol	Elective	**913**
RLP	Resource Location Protocol	Elective	**887**
SUPDUP	SUPDUP Protocol	Elective	**734**

Applicability Statements:

IP-SMDS and IP-ARCNET -- These define methods of sending IP over particular network types. It is anticipated that these will be advanced to the network specific standard protocol state in the future.

Telnet Options

For convenience all the Telnet Options are collected here with both their state and status.

Protocol	Name	Number	State	Status	RFC
TOPT-BIN	Binary Transmission	0	Std	Rec	**856**
TOPT-ECHO	Echo	1	Std	Rec	**857**
TOPT-RECN	Reconnection	2	Prop	Ele	
TOPT-SUPP	Suppress Go Ahead	3	Std	Rec	**858**
TOPT-APRX	Approx Message Size Negotiation	4	Prop	Ele	
TOPT-STAT	Status	5	Std	Rec	**859**
TOPT-TIM	Timing Mark	6	Std	Rec	**860**
TOPT-REM	Remote Controlled Trans and Echo	7	Prop	Ele	**726**
TOPT-OLW	Output Line Width	8	Prop	Ele	
TOPT-OPS	Output Page Size	9	Prop	Ele	
TOPT-OCRD	Output Carriage-Return Disposition	10	Prop	Ele	**652**
TOPT-OHT	Output Horizontal Tabstops	11	Prop	Ele	**653**
TOPT-OHTD	Output Horizontal Tab Disposition	12	Prop	Ele	**654**
TOPT-OFD	Output Formfeed Disposition	13	Prop	Ele	**655**
TOPT-OVT	Output Vertical Tabstops	14	Prop	Ele	**656**
TOPT-OVTD	Output Vertical Tab Disposition	15	Prop	Ele	**657**
TOPT-OLD	Output Linefeed Disposition	16	Prop	Ele	**658**
TOPT-EXT	Extended ASCII	17	Prop	Ele	**698**
TOPT-LOGO	Logout	18	Prop	Ele	**727**
TOPT-BYTE	Byte Macro	19	Prop	Ele	**735**
TOPT-DATA	Data Entry Terminal	20	Prop	Ele	**1043**
TOPT-SUP	SUPDUP	21	Prop	Ele	**734**
TOPT-SUPO	SUPDUP Output	22	Prop	Ele	**749**
TOPT-SNDL	Send Location	23	Prop	Ele	**779**
TOPT-TERM	Terminal Type	24	Prop	Ele	**930**
TOPT-EOR	End of Record	25	Prop	Ele	**885**
TOPT-TACACS	TACACS User Identification	26	Prop	Ele	**927**
TOPT-OM	Output Marking	27	Prop	Ele	**933**
TOPT-TLN	Terminal Location Number	28	Prop	Ele	**946**
TOPT-3270	Telnet 3270 Regime	29	Prop	Ele	**1041**
TOPT-X.3	X.3 PAD	30	Prop	Ele	**1053**
TOPT-NAWS	Negotiate About Window Size	31	Prop	Ele	**1073**
TOPT-TS	Terminal Speed	32	Prop	Ele	**1079**
TOPT-RFC	Remote Flow Control	33	Prop	Ele	**1080**
TOPT-LINE	Linemode	34	Draft	Ele	**1184**
TOPT-XDL	X Display Location	35	Prop	Ele	**1096**
TOPT-EXTOP	Extended-Options-List	255	Std	Rec	**861**

RFC-1250 IAB Official Protocol Standards August 1991; The Protocols

Experimental Protocols

Protocol	Name	Status	RFC
IN-ENCAP	Internet Encapsulation Protocol	Limited Use	**1241***
CLNS-MIB	CLNS-MIB	Limited Use	**1238***
CFDP	Coherent File Distribution Protocol	Limited Use	**1235***
SNMP-DPI	SNMP Distributed Program Interface	Limited Use	**1228***
SNMP-MUX	SNMP MUX Protocol and MIB	Limited Use	**1227***
IP-AX25	IP Encapsulation of AX.25 Frames	Limited Use	**1226***
ALERTS	Managing Asynchronously Generated Alerts	Limited Use	**1224**
MPP	Message Posting Protocol	Limited Use	**1204**
ST-II	Stream Protocol	Limited Use	**1190**
SNMP-BULK	Bulk Table Retrieval with the SNMP	Limited Use	**1187**
DNS-RR	New DNS RR Definitions	Limited Use	**1183**
NTP-OSI	NTP over OSI Remote Operations	Limited Use	**1165**
MSP	Message Send Protocol	Limited Use	**1159**
EHF-MAIL	Encoding Header Field for Mail	Elective	**1154**
DMF-MAIL	Digest Message Format for Mail	Elective	**1153**
RDP	Reliable Data Protocol	Limited Use	**908**,**1151**
--------	Mapping between X.400(88) and RFC-822	Elective	**1148**
TCP-ACO	TCP Alternate Checksum Option	Not Recom.	**1146**
--------	Mapping full 822 to Restricted 822	Elective	**1137**
IP-DVMRP	IP Distance Vector Multicast Routing	Not Recom.	**1075**
TCP-LDP	TCP Extensions for Long Delay Paths	Limited Use	**1072**
IMAP2	Interactive Mail Access Protocol	Limited Use	**1176**,**1064**
IMAP3	Interactive Mail Access Protocol	Limited Use	**1203**
VMTP	Versatile Message Transaction Protocol	Elective	**1045**
COOKIE-JAR	Authentication Scheme	Not Recom.	**1004**
NETBLT	Bulk Data Transfer Protocol	Not Recom.	**998**
IRTP	Internet Reliable Transaction Protocol	Not Recom.	**938**
AUTH	Authentication Service	Not Recom.	**931**
LDP	Loader Debugger Protocol	Not Recom.	**909**
NVP-II	Network Voice Protocol	Limited Use	ISI-memo
PVP	Packet Video Protocol	Limited Use	ISI-memo

RFC-1250 IAB Official Protocol Standards August 1991; The Protocols

Informational Protocols

Protocol	Name	Status	RFC
DIXIE	DIXIE Protocol Specification	Limited Use	**1249***
IP-X.121	IP to X.121 Address Mapping for DDN	Limited Use	**1236***
OSI-HYPER	OSI and LLC1 on HYPERchannel	Limited Use	**1223***
HAP2	Host Access Protocol	Limited Use	**1221***
SUBNETASGN	On the Assignment of Subnet Numbers	Limited Use	**1219***
SNMP-TRAPS	Defining Traps for use with SNMP	Limited Use	**1215**
DAS	Directory Assistance Service	Limited Use	**1202**
MD4	MD4 Message Digest Algorithm	Limited Use	**1186**
LPDP	Line Printer Daemon Protocol	Limited Use	**1179**

RFC-1250 IAB Official Protocol Standards August 1991; The Protocols

Historic Protocols

Protocol	Name	Status	RFC	
SGMP	Simple Gateway Monitoring Protocol	NR	**1028**	
HEMS	High Level Entity Management Protocol	NR	**1021**	
STATSRV	Statistics Server	NR	**996**	
POP2	Post Office Protocol, Version 2	NR	**937**	
RATP	Reliable Asynchronous Transfer Protocol	NR	**916**	
HFEP	Host - Front End Protocol	NR	**929***	
THINWIRE	Thinwire Protocol	NR	**914**	
HMP	Host Monitoring Protocol	NR	**869**	
GGP	Gateway Gateway Protocol	NR	**823**	
RTELNET	Remote Telnet Service	NR	**818**	
CLOCK	DCNET Time Server Protocol	NR	**778**	
MPM	Internet Message Protocol	NR	**759**	
NETRJS	Remote Job Service	NR	**740**	
NETED	Network Standard Text Editor	NR	**569**	
RJE	Remote Job Entry	NR	**407**	
XNET	Cross Net Debugger	NR	IEN-158	
NAMESERVER	Host Name Server Protocol	NR	IEN-116	
MUX	Multiplexing Protocol	NR	IEN-90	
GRAPHICS	Graphics Protocol	NR	NIC-24308	

Contacts

Internet Contacts

Internet Activities Board (IAB) Contact

Bob Braden
Executive Director of the IAB
USC/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292-6695

1-213-822-1511
Braden@ISI.EDU

Please send your comments about this list of protocols and especially about the Draft Standard Protocols to the Internet Activities Board care of Bob Braden, IAB Executive Director.

Internet Engineering Task Force (IETF) Contact

Phill Gross
Chair of the IETF
Corporation for National Research Initiatives (NRI)
1895 Preston White Drive, Suite 100
Reston, VA 22091

1-703-620-8990
PGross@NRI.RESTON.VA.US

Greg Vaudreuil
IESG Secretary
Corporation for National Research Initiatives
1895 Preston White Drive, Suite 100
Reston, VA 22091

1-703-620-8990
gvaudre@NRI.RESTON.VA.US

Internet Research Task Force (IRTF) Contact

David D. Clark
Chair of the IRTF
Massachusetts Institute of Technology
Laboratory for Computer Science
545 Main Street
Cambridge, MA 02139

1-617-253-6003
ddc@LCS.MIT.EDU

Internet Assigned Numbers Authority Contact

Joyce K. Reynolds
Internet Assigned Numbers Authority
USC/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292-6695

1-213-822-1511
IANA@ISI.EDU

The protocol standards are managed for the IAB by the Internet Assigned Numbers Authority.

Please refer to the documents "Assigned Numbers" (RFC-1060) and "Official Internet Protocols" (RFC-1011) for further information about the status of protocol documents. There are two documents that summarize the requirements for host and gateways in the Internet, "Host Requirements" (RFC-1122 and RFC-1123) and "Gateway Requirements" (RFC-1009).

Note: The material in RFC-1122 and RFC-1123 has been integrated into the original RFCs contained in this system, and therefore they are not explicitly included.

How to obtain the most recent edition of this "IAB Official Protocol Standards" memo:

> The file "in-notes/iab-standards.txt" may be copied via FTP from the VENERA.ISI.EDU computer using the FTP username "anonymous" and FTP password "guest".

Request for Comments Editor Contact

Jon Postel
RFC Editor
USC/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292-6695

1-213-822-1511
Postel@ISI.EDU

Documents may be submitted via electronic mail to the RFC Editor for consideration for publication as RFC. If you are not familiar with the format or style requirements please request the "Instructions for RFC Authors". In general, the style of any recent RFC may be used as a guide.

The Network Information Center
and
Requests for Comments Distribution Contact

DDN Network Information Center
SRI International
Room EJ291
333 Ravenswood Avenue
Menlo Park, CA 94025

1-800-235-3155
1-415-859-3695

NIC@NIC.DDN.MIL

The Network Information Center (NIC) provides many information services for the Internet community. Among them is maintaining the Requests for Comments (RFC) library.

RFCs can be obtained via FTP from NIC.DDN.MIL, with the pathname RFC:RFCnnnn.TXT where "nnnn" refers to the number of the RFC. A list of all RFCs may be obtained by copying the file RFC:RFC-INDEX.TXT. Log in with FTP username ANONYMOUS and password GUEST.

The NIC also provides an automatic mail service for those sites which cannot use FTP. Address the request to SERVICE@NIC.DDN.MIL and in the subject field of the message indicate the file name, as in

"Subject: SEND RFC:RFCnnnn.TXT".

Some RFCs are now available in PostScript, these may be obtained from the NIC in a similar fashion by substituting ".PS" for ".TXT".

How to obtain the most recent edition of this "IAB Official Protocol Standards" memo:

The file RFC:IAB-STANDARDS.TXT may be copied via FTP from the NIC.DDN.MIL computer following the same procedures used to obtain RFCs.

Other Sources for Requests for Comments

NSF Network Service Center (NNSC)

NSF Network Service Center (NNSC)
BBN Laboratories, Inc.
10 Moulton St.
Cambridge, MA 02238

617-873-3400

NNSC@NNSC.NSF.NET

NSF Network Information Service (NIS)

NSF Network Information Service
Merit Computer Network
University of Michigan
1075 Beal Avenue
Ann Arbor, MI 48109

313-763-4897

INFO@NIS.NSF.NET

CSNET Coordination and Information Center (CIC)

CSNET Coordination and Information Center
BBN Systems and Technologies Corporation
10 Moulton Street
Cambridge, MA 02238

617-873-2777

INFO@SH.CS.NET

Author's Address

Jon Postel
USC/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292

Phone: (213) 822-1511

Email: Postel@ISI.EDU

RFC-652 Telnet Output Carriage-Return Disposition Option

D. Crocker UCLA-NMC
25 October 1974

Command name and code

NAOCRD 10 (Negotiate About Output Carriage-Return Disposition)

Default

DON'T NAOCRD/WON'T NAOCRD.

In the default absence of negotiations concerning which party, data sender or data receiver, is handling output carriage-returns, neither party is required to handle carriage-returns and neither party is prohibited from handling them; but it is appropriate if at least the data receiver handles carriage-returns, albeit primitively.

Command Meanings
Motivation
Description

Command Meanings

In the following, we are discussing a simplex connection, as described in the NAOL and NAOP Telnet options.

IAC DO NAOCRD

The data sender requests or agrees to negotiate about output carriage-return character disposition with the data receiver. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver is assumed to be handling output carriage-returns.

IAC DON'T NAOCRD

The data sender refuses to negotiate about output carriage-return disposition with the data receiver, or demands a return to the unnegotiated default mode.

IAC WILL NAOCRD

The data receiver requests or agrees to negotiate about output carriage-return disposition with the sender. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver alone is assumed to be handling output carriage-returns.

IAC WON'T NAOCRD

The data receiver refuses to negotiate about output carriage-return disposition, or demands a return to the unnegotiated default mode.

IAC SB NAOCRD DS <8-bit value> IAC SE

The data sender specifies, with the 8-bit value, which party should handle carriage-returns and what their disposition should be. The code for DS is 1.

IAC SB NAOCRD DR <8-bit value> IAC SE

The data receiver specifies, with the 8-bit value, which party should handle carriage-returns and what their disposition should be. The code for DR is 0.

RFC-652 Telnet Output Carriage-Return Disposition Option

Motivation for the Option

There appear to be four cases in which it is useful for the party at one end of a TELNET connection to communicate with the other party about carriage-return disposition:

a The sender may wish the receiver to use its local knowledge of the terminal to properly handle carriage-returns;

b The receiver may wish the sender to use its local knowledge of the terminal to properly handle carriage-returns;

c The sender may wish to use its local knowledge of the data being sent to instruct the receiver in the proper handling of carriage-returns;

d The receiver may wish to use its local knowledge of the terminal to instruct the sender in the proper handling of carriage-returns.

RFC-652 Telnet Output Carriage-Return Disposition Option

Description of the Option

The data sender and the data receiver use the 8-bit value along with the NAOCRD SB commands as follows:

8-bit value	Meaning
0	Command sender suggests that he alone will handle carriage-returns, for the connection.
1 to 250	Command sender suggests that the other party alone should handle carriage-returns, but suggests that a delay of the indicated value be used. The value is the number of character-times to wait or number of NULs to insert in the data stream before sending the next data character. (See qualification, below.)
251	Not allowed, in order to be compatible with related Telnet options.
252	Command sender suggests that the other party alone handle carriage-returns, but suggests that they be discarded.
253	Not allowed, in order to be compatible with related Telnet options.
254	Command sender suggests that the other party alone should handle carriage-returns but suggests waiting for a character to be transmitted (on the other simplex connection) before sending more data. (See qualification, below.) Note that, due to the assynchrony of the two simplex connections, phase problems can occur with this option.
255	Command sender suggests that the other party alone should handle carriage-returns and suggests nothing about how it should be done.

The guiding rules are that:

(1) if neither data receiver nor data sender wants to handle carriage-returns, the data receiver must do it, and

(2) if both data receiver and data sender want to handle carriage-returns, the data sender gets to do it.

The reasoning for the former rule is that if neither wants to do it, then the default in the NAOCRD option dominates. If both want to do it, the sender, who is presumed to have special knowledge about the data, should be allowed to do it, taking into account any suggestions the receiver may make.

Note that carriage-return delays, controlled by the data sender, must consist of NUL characters inserted immediately after the character in question. This is necessary due to the assynchrony of network transmissions. Due to the Telnet end-of-line convention, with carriage-returns followed by a linefeed, any NULs that would otherwise be placed after the carriage-return must be placed after the linefeed, regardless of any modifications that may additionally be made to the line feed (see NAOLFD Telnet option).

As with all option negotiations, neither party should suggest a state already in effect except to refuse to negotiate; changes should be acknowledged; and once refused, an option should not be resuggested until "something changes" (e.g., another process starts).

At any time, either party can disable further negotiation by giving the appropriate WON'T NAOCRD or DON'T NAOCRD command.

RFC-653 Telnet Output Horizontal Tabstops Option

D. Crocker UCLA-NMC
25 October 1974

Command name and code

NAOHTS 11 Negotiate About Output Horizontal Tabstops

Default

DON'T NAOHTS/WON'T NAOHTS.

In the default absence of negotiations concerning which party, data sender or data receiver, is handling output horizontal tabstops, neither party is required to handle them and neither party is prohibited from handling them; but it is appropriate if at least the data receiver handles horizontal tabstops, albeit primitively.

Command Meanings
Motivation
Description

Command Meanings

In the following, we are discussing a simplex connection, as described in the NAOL and NAOP Telnet options.

IAC DO NAOHTS

The data sender requests or agrees to negotiate about output horizontal tabstops with the data receiver. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver is assumed to be handling output horizontal tabstops.

IAC DON'T NAOHTS

The data sender refuses to negotiate about output horizontal tabstops with the data receiver, or demands a return to the unnegotiated default mode.

IAC WILL NAOHTS

The data receiver requests or agrees to negotiate about output horizontal tabstops with the sender. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver alone is assumed to be handling output horizontal tabstops.

IAC WON'T NAOHTS

The data receiver refuses to negotiate about output horizontal tabstops, or demands a return to the unnegotiated default mode.

IAC SB NAOHTS DS <8-bit value> ... <8-bit value> IAC SE

The data sender specifies, with the 8-bit value(s), which party should handle output horizontal tabstop considerations and what the stops should be. The code for DS is 1.

IAC SB NAOHTS DR <8-bit value> ... <8-bit value> IAC SE

The data receiver specifies, with the 8-bit value(s), which party should handle output horizontal tabstop considerations and what the stops should be. The code for DR is 0.

RFC-653 Telnet Output Horizontal Tabstops Option

Motivation for the Option

There appear to be four cases in which it is useful for the party at one end of a TELNET connection to communicate with the other party about horizontal tabstop disposition:

a The sender may wish the receiver to use its local knowledge of the terminal to properly handle horizontal tabstops;

b The receiver may wish the sender to use its local knowledge of the terminal to properly handle horizontal tabstops;

c The sender may wish to use its local knowledge of the data being sent to instruct the receiver in the proper handling of horizontal tabstops;

d The receiver may wish to use its local knowledge of the terminal to instruct the sender in the proper handling of horizontal tabstops.

RFC-653 Telnet Output Horizontal Tabstops Option

Description of the Option

The data sender and the data receiver use the 8-bit value(s) along with the DS and DR SB subcommands as follows (multiple 8-bit values are allowed only if each is greater than zero and less than 251):

8-bit value:	Meaning:
0	Command sender suggests that he alone will handle tabstops, for the connection.
1 to 250	Command sender suggests that the other party alone should handle tabstop considerations, but suggests that the indicated value(s) be used. The value(s) are the column numbers, relative to the physical left side of the printer page or terminal screen, that are to be set.
251 to 254	Not allowed, in order to be compatible with related Telnet options.
255	Command sender suggests that the other party alone should handle output tabstops and suggests nothing about how it should be done.

The guiding rules are that:

(1) if neither data receiver nor data sender wants to handle output horizontal tabstops, the data receiver must do it, and

(2) if both data receiver and data sender want to handle output horizontal tabstops, the data sender gets to do it.

The reasoning for the former rule is that if neither wants to do it, then the default in the NAOHTS option dominates. If both want to do it, the sender, who is presumed to have special knowledge about the data, should be allowed to do it, taking into account any suggestions the receiver may make. As with all option negotiations, neither party should suggest a state already in effect except to refuse to negotiate; changes should be acknowledged; and once refused, an option should not be resuggested until "something changes" (e.g., another process starts). At any time, either party can disable further negotiation by giving the appropriate WON'T NAOHTS or DON'T NAOHTS command.

RFC-654 Telnet Output Horizontal Tab Disposition Option

D. Crocker UCLA-NMC
25 October 1974

Command name and code

NAOHTD 12 Negotiate About Output Horizontal Tab Disposition

Default

DON'T NAOHTD/WON'T NAOHTD.

In the default absence of negotiations concerning which party, data sender or data receiver, is handling output horizontal tab character considerations, neither party is required to handle horizontal tab characters and neither party is prohibited from handling them; but it is appropriate if at least the data receiver handles horizontal tab character considerations, albeit primitively.

RFC-654 Telnet Output Horizontal Tab Disposition Option

Command meanings

In the following, we are discussing a simplex connection, as described in the NAOL and NAOP Telnet options.

IAC DO NAOHTD

The data sender requests or agrees to negotiate about output horizontal tab character disposition with the data receiver. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver is assumed to be handling output horizontal tab character considerations.

IAC DON'T NAOHTD

The data sender refuses to negotiate about output horizontal tab characters with the data receiver, or demands a return to the unnegotiated default mode.

IAC WILL NAOHTD

The data receiver requests or agrees to negotiate about output horizontal tab characters with the sender. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver alone is assumed to be handling output horizontal tab character considerations.

IAC WON'T NAOHTD

The data receiver refuses to negotiate about output horizontal tab characters, or demands a return to the unnegotiated default mode.

IAC SB NAOHTD DS <8-bit value> IAC SE

The data sender specifies, with the 8-bit value, which party should handle output horizontal tab characters and what their disposition should be. The code for DS is 1.

IAC SB NAOHTD DR <8-bit value> IAC SE

The data receiver specifies, with the 8-bit value, which party should handle output horizontal tab characters and what their disposition should be. The code for DR is 0.

RFC-654 Telnet Output Horizontal Tab Disposition Option

Motivation for the Option

There appear to be four cases in which it is useful for the party at one end of a TELNET connection to communicate with the other party about horizontal tabstop disposition:

a The sender may wish the receiver to use its local knowledge of the terminal to properly handle horizontal tabstops;

b The receiver may wish the sender to use its local knowledge of the terminal to properly handle horizontal tabstops;

c The sender may wish to use its local knowledge of the data being sent to instruct the receiver in the proper handling of horizontal tabstops;

d The receiver may wish to use its local knowledge of the terminal to instruct the sender in the proper handling of horizontal tabstops.

RFC-654 Telnet Output Horizontal Tab Disposition Option

Description of the Option

The data sender and the data receiver use the 8-bit value along with the DS and DR SB commands as follows:

8-bit value	Meaning
0	Command sender suggests that he alone will handle horizontal tab characters, for the connection.
1 to 250	Command sender suggests that the other party alone should handle horizontal tab characters, but suggests that a delay of the indicated value be used. The value is the number of character-times to wait or number of NULs to insert in the data stream before sending the next data character.
251	Command sender suggests that the other party alone handle horizontal tabs, but suggests that each occurrence of the character be replaced by a space.
252	Command sender suggests that the other party alone handle horizontal tabs, but suggests that they be discarded.
253	Command sender suggests that the other party alone should handle horizontal tab characters, but suggests that tabbing be simulated.
254	Command sender suggests that the other party alone should handle horizontal tab characters, but suggests that waiting for a character to be transmitted (on the other simplex connection) before sending more data. Note that, due to the assynchrony of the two simplex connections, phase problems can occur with this option.
255	Command sender suggests that the other party alone should handle output horizontal tabs and suggests nothing about how it should be done.

The guiding rules are that:

1) if neither data receiver nor data sender wants to handle output horizontal tab characters, the data receiver must do it, and

2) if both data receiver and data sender wants to handle output horizontal tab characters, the data sender gets to do it.

The reasoning for the former rule is that if neither wants to do it, then the default in the NAOHTD option dominates. If both want to do it, the sender, who is presumed to have special knowledge about the data, should be allowed to do it, taking into account any suggestions the receiver may make. Simulation is defined as the replacement of the horizontal tab character by enough spaces to move the printer head (or line-pointer) to the next horizontal tab stop.

Note that delays, controlled by the data sender, must consist of NUL characters inserted immediately after the horizontal tab character. This is necessary due to the assynchrony of network transmissions. As with all option negotiations, neither party should suggest a state already in effect except to refuse to negotiate; changes should be acknowledged; and once refused, an option should not be resuggested until "something changes" (e.g., another process starts). At any time, either party can disable further negotiation by giving the appropriate WON'T NAOHTD or DON'T NAOHTD command.

RFC-655 Telnet Output Formfeed Disposition Option

D. Crocker UCLA-NMC
25 October 1974

Command name and code

NAOFFD - 13 Negotiate About Output Formfeed Disposition

Default

DON'T NAOFFD/WON'T NAOFFD

In the default absence of negotiations concerning which party, data sender or data receiver, is handling output formfeeds, neither party is required to handle formfeeds and neither party is prohibited from handling them; but it is appropriate if at least the data receiver handles formfeed considerations, albeit primitively.

Command Meanings

In the following, we are discussing a simplex connection, as described in the NAOL and NAOP Telnet Options specifications.

IAC DO NAOFFD

The data sender requests or agrees to negotiate about output formfeed disposition with the data receiver. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver is assumed to be handling output formfeeds.

IAC DON'T NAOFFD

The data sender refuses to negotiate about output formfeed disposition with the data receiver, or demands a return to the unnegotiated default mode.

IAC WILL NAOFFD

The data receiver requests or agrees to negotiate about output formfeed disposition with the sender. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver alone is assumed to be handling output formfeeds.

IAC WON'T NAOFFD

The data receiver refuses to negotiate about output formfeed disposition, or demands a return to the unnegotiated default mode.

IAC SB NAOFFD DS <8-bit value> IAC SE

The data sender specifies, with the 8-bit value, which party should handle formfeeds and what their disposition should be. The code for DS is 1.

IAC SB NAOFFD DR <8-bit value> IAC SE

The data receiver specifies, with the 8-bit value, which party should handle formfeeds and what their disposition should be. The code for DR is 0.

RFC-655 Telnet Output Formfeed Disposition Option

Motivation for the Option

There appear to be four cases in which it is useful for the party at one end of a TELNET connection to communicate with the other party about formfeed disposition:

a The sender may wish the receiver to use its local knowledge of the terminal to properly handle formfeed;

b The receiver may wish the sender to use its local knowledge of the terminal to properly handle formfeeds;

c The sender may wish to use its local knowledge of the data being sent to instruct the receiver in the proper handling of formfeeds;

d The receiver may wish to use its local knowledge of the terminal to instruct the sender in the proper handling of formfeeds.

RFC-655 Telnet Output Formfeed Disposition Option

Description of the Option

The data sender and the data receiver use the 8-bit value along with the DS and DR SB commands as follows:

8-bit value	Meaning
0	Command sender suggests that he alone will handle formfeeds, for the connection.
1 to 250	Command sender suggests that the other party alone should handle formfeeds, but suggests that the indicated value be used. The value is the number of character-times to wait or number of NULs to insert in the data stream before sending the next data character.
251	Command sender suggests that the other party alone handle formfeeds, but suggests that each occurrence of the character be replaced by carriage-return/line-feed.
252	Command sender suggests that the other party alone handle formfeeds, but suggests that they be discarded.
253	Command sender suggests that the other party alone should handle formfeeds, but suggests that formfeeds be simulated.
254	Command sender suggests that the other party alone should handle output formfeeds but suggests waiting for a character to be transmitted (on the other simplex connection) before sending more data. Note that, due to the assynchrony of the two simplex connections, phase problems can occur with this option.
255	Command sender suggests that the other party alone should handle output formfeeds and suggests nothing about how it should be done.

The guiding rules are that:

1) if neither data receiver nor data sender wants to handle output formfeeds, the data receiver must do it, and

2) if both data receiver and data sender want to handle output formfeeds, the data sender gets to do it.

The reasoning for the former rule is that if neither wants to do it, then the default in the NAOFFD option dominates. If both want to do it, the sender, who is presumed to have special knowledge about the data, should be allowed to do it, taking into account any suggestions the receiver may make. Simulation is defined as the replacement of the formfeed character by enough line-feeds (only) to advance the paper (or line-pointer) to the top of the next page (or to the top of the terminal screen). Note that delays, controlled by the data sender, must consist of NUL characters inserted immediately after the formfeed character. This is necessary due to the assynchrony of network transmission. As with all option negotiations, neither party should suggest a state already in effect except to refuse to negotiate; changes should be acknowledged; and once refused, an option should not be resuggested until "something changes" (e.g., another process starts). At any time, either party can disable further negotiation by giving the appropriate WON'T NAOFFD or DON'T NAOFFD command.

RFC-656 Telnet Output Vertical Tabstops Option

D. Crocker UCLA-NMC
25 October 1974

Command name and code

NAOVTS 14 Negotiate About Vertcial Tabstops

Default

DON'T NAOVTS/WON'T NAOVTS.

In the default absence of negotiations concerning which party, data sender or data receiver, is handling output vertical tabstop considerations, neither party is required to handle vertical tabstops and neither party is prohibited from handling them; but it is appropriate if at least the data receiver handles vertical tabstop considerations, albeit primitively.

Command Meanings

In the following, we are discussing a simplex connection, as described in the NAOL and NAOP Telnet Options specifications.

IAC DO NAOVTS

The data sender requests or agrees to negotiate about output vertical tabstops with the data receiver. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver is assumed to be handling output vertical tabstop considerations.

IAC DON'T NAOVTS

The data sender refuses to negotiate about output vertical tabstops with the data receiver, or demands a return to the unnegotiated default mode.

IAC WILL NAOVTS

The data receiver requests or agrees to negotiate about output vertical tabstops with the sender. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver alone is assumed to be handling output vertical tabstop considerations.

IAC WON'T NAOVTS

The data receiver refuses to negotiate about output vertical tabstops, or demands a return to the unnegotiated default mode.

IAC SB NAOVTS DS <8-bit value> ... <8-bit value> IAC SE

The data sender specifies, with the 8-bit value(s), which party should handle output vertical tabstop considerations and what the stops should be. The code for DS is 1.

IAC SB NAOVTS DR <8-bit value> ... <8-bit value> IAC SE

The data receiver specifies, with the 8-bit value(s), which party should handle output vertical tabstop considerations and what the stops should be. The code for DR is 0.

RFC-656 Telnet Output Vertical Tabstops Option

Motivation for the Option

There appear to be four cases in which it is useful for the party at one end of a TELNET connection to communicate with the other party about vertical tabstop disposition:

a The sender may wish the receiver to use its local knowledge of the terminal to properly handle vertical tabstops;

b The receiver may wish the sender to use its local knowledge of the terminal to properly handle vertical tabstops;

c The sender may wish to use its local knowledge of the data being sent to instruct the receiver in the proper handling of vertical tabstops;

d The receiver may wish to use its local knowledge of the terminal to instruct the sender in the proper handling of vertical tabstops.

RFC-656 Telnet Output Vertical Tabstops Option

Description of the Option

The data sender and the data receiver use the 8-bit value(s) along with the DS and DR SB commands as follows (multiple 8-bit values are allowed only if each is greater than zero and less than 251):

8-bit value	Meaning
0	Command sender suggests that he alone will handle the vertical tabstops, for the connection.
1 to 250	Command sender suggests that the other party alone should handle the stops, but suggests that the indicated value(s) be used. Each value is the line number, relative to the top of the printer page or terminal screen, that is to be set as a vertical tabstop.
251 to 254	Not allowed, in order to be compatible with related Telnet options.
255	Command sender suggests that the other party alone should handle output vertical tabstops and suggests nothing about how it should be done.

The guiding rules are that:

1) if neither data receiver nor data sender wants to handle output vertical tabstops, the data receiver must do it, and
2) if both data receiver and data sender want to handle output vertical tabstops, the data sender gets to do it.

The reasoning for the former rule is that if neither wants to do it, then the default in the NAOVTS option dominates. If both want to do it, the sender, who is presumed to have special knowledge about the data, should be allowed to do it, taking into account any suggestions the receiver may make. This is necessary due to the assynchrony of network transmissions. As with all option negotiations, neither party should suggest a state already in effect except to refuse to negotiate; changes should be acknowledged; and once refused, an option should not be resuggested until "something changes" (e.g., another process starts). At any time, either party can disable further negotiation by giving the appropriate WON'T NAOVTS or DON'T NAOVTS command.

RFC-657 Telnet Output Vertical Tab Disposition Option

D. Crocker UCLA-NMC
25 October 1974

Command name and code

NAOVTD 15 Negotiate About Output Vertcial Tab Disposition

Default

DON'T NAOVTD/WON'T NAOVTD

In the default absence of negotiations concerning which party, data sender or data receiver, is handling output vertical tab character considerations, neither party is required to handle vertical tab characters and neither party is prohibited from handling them; but it is appropriate if at least the data receiver handles vertical tab character considerations, albeit primitively.

RFC-657 Telnet Output Vertical Tab Disposition Option

Command meanings

In the following, we are discussing a simplex connection, as described in the NAOL and NAOP Telnet Options specifications.

IAC DO NAOVTD

The data sender requests or agrees to negotiate about output vertical tab character disposition with the data receiver. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver is assumed to be handling output vertical tab character considerations.

IAC DON'T NAOVTD

The data sender refuses to negotiate about output vertical tab character disposition with the data receiver, or demands a return to the unnegotiated default mode.

IAC WILL NAOVTD

The data receiver requests or agrees to negotiate about output vertical tab character disposition with the sender. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver alone is assumed to be handling output vertical tab character considerations.

IAC WON'T NAOVTD

The data receiver refuses to negotiate about output vertical tab character disposition, or demands a return to the unnegotiated default mode.

IAC SB NAOVTD DS <8-bit value> IAC SE

The data sender specifies, with the 8-bit value, which party should handle output vertical tab characters and what their disposition should be. The code for DS is 1.

IAC SB NAOVTD DR <8-bit value> IAC SE

The data receiver specifies, with the 8-bit value, which party should handle output vertical tab characters and what their disposition should be. The code for DR is 0.

RFC-657 Telnet Output Vertical Tab Disposition Option

Motivation for the Option

There appear to be four cases in which it is useful for the party at one end of a TELNET connection to communicate with the other party about vertical tabstop disposition:

a The sender may wish the receiver to use its local knowledge of the terminal to properly handle vertical tabstops;

b The receiver may wish the sender to use its local knowledge of the terminal to properly handle vertical tabstops;

c The sender may wish to use its local knowledge of the data being sent to instruct the receiver in the proper handling of vertical tabstops;

d The receiver may wish to use its local knowledge of the terminal to instruct the sender in the proper handling of vertical tabstops.

RFC-657 Telnet Output Vertical Tab Disposition Option

Description of the Option

The data sender and the data receiver use the 8-bit value along with the DS and DR SB commands as follows:

8 bit value	Meaning
0	Command sender suggests that he alone will handle vertical tab characters, for the connection.
1 to 250	Command sender suggests that the other party alone should handle tab characters, but suggests that a delay of the indicated value be used. The value is the number of character-times to wait or number of NULs to insert in the data stream before sending the next data character.
251	Command sender suggests that the other party alone handle vertical tabs, but suggests that each occurrence of the character be replaced by carriage-return/linefeed.
252	Command sender suggests that the other party alone handle vertical tabs, but suggests that they be discarded.
253	Command sender suggests that the other party alone should handle tab characters, but suggests that tabbing be simulated.
254	Command sender suggests that the other party alone should handle the output disposition but suggests waiting for a character to be transmitted (on the other simplex connection) before sending more data. Note that, due to the assynchrony of the two simplex connections, phase problems can occur with this option.
255	Command sender suggests that the other party alone hould handle the output disposition and suggests nothing about how it should be done.

The guiding rules are that:

1. if neither data receiver nor data sender wants to handle the output vertical tab characters, the data receiver must do it, and
2. if both data receiver and data sender want to handle the output vertical tab characters, the data sender gets to do it.

The reasoning for the former rule is that if neither want to do it, then the default in the NAOVTD option dominates. If both want to do it, the sender, who is presumed to have special knowledge about the data, should be allowed to do it, taking into account any suggestions the receiver may make. Simulation is defined as the replacement of the character by enough line-feeds (only) to advance the paper (or line-pointer) to the next vertical tab stop. Note that delays, controlled by the data sender, must consist of NUL characters, inserted immediately after the line-feed character. This is necessary due to the assynchrony of network transmissions. As with all option negotiations, neither party should suggest a state already in effect except to refuse to negotiate; changes should be acknowledged; and once refused, an option should not be resuggested until "something changes" (e.g., another process starts). At any time, either party can disable further negotiation by giving the appropriate WON'T NAOVTD or DON'T NAOVTD command.

RFC-658 Telnet Output Linefeed Disposition

D. Crocker UCLA-NMC
25 October 1974

Command name and code

NAOLFD 16 Negotiate About Output Linefeed Disposition

Default

DON'T NAOLFD/WON'T NAOLFD.

In the default absence of negotiations concerning which party, data under or data receiver, is handling output linefeed considerations, neither party is required nor prohibited from handling linefeeds; but it is appropriate if at least the data receiver handles them, albeit primitively.

Command Meanings
Motivation
Description

Command meanings

In the following, we are discussing a simplex connection, as described in the NAOL and NAOP Telnet Options.

IAC DO NAOLFD

The data sender requests or agrees to negotiate about output linefeed disposition with the data receiver. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver is assumed to be handling output linefeed considerations.

IAC DON'T NAOLFD

The data sender refuses to negotiate about output linefeed disposition with the data receiver, or demands a return to the unnegotiated default mode.

IAC WILL NAOLFD

The data receiver requests or agrees to negotiate about output linefeed disposition with the sender. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver alone is assumed to be handling output linefeed considerations.

IAC WON'T NAOLFD

The data receiver refuses to negotiate about output linefeed disposition, or demands a return to the unnegotiated default mode.

IAC SB NAOLFD DS <8-bit value> IAC SE

The data sender specifies, with the 8-bit value, which party should handle output linefeeds and what their disposition should be. The code for DS is 1.

IAC SB NAOLFD DR <8-bit value> IAC SE

The data receiver specifies, with the 8-bit value, which party should handle output linefeeds and what their disposition should be. The code for DR is 0.

RFC-658 Telnet Output Linefeed Disposition Option

Motivation for the Option

There appear to be four cases in which it is useful for the party at one end of a TELNET connection to communicate with the other party about linefeed disposition:

a The sender may wish the receiver to use its local knowledge of the terminal to properly handle linefeeds;

b The receiver may wish the sender to use its local knowledge of the terminal to properly handle linefeeds;

c The sender may wish to use its local knowledge of the data being sent to instruct the receiver in the proper handling of linefeeds;

d The receiver may wish to use its local knowledge of the terminal to instruct the sender in the proper handling of linefeeds.

RFC-658 Telnet Output Linefeed Disposition Option

Description of the Option

The data sender and the data receiver use the 8-bit value along with DS and DR SB commands as follows:

8-bit value	Meaning
0	Command sender suggests that he alone will handle linefeeds, for the connection.
1 to 250	Command sender suggests that the other party alone should handle linefeeds, but suggests that a delay of the indicated value be used. The value is the number of character-times to wait or number of NULs to insert in the data stream before sending the next data character. (See qualifications, below.)
251	Not allowed, in order to be compatible with related Telnet options.
252	Command sender suggests that the other party alone handle linefeeds, but suggests that they be discarded.
253	Command sender suggests that the other party alone should handle linefeeds, but suggests that linefeeds be simulated.
254	Command sender suggests that the other party alone should handle output linefeeds but suggests waiting for a character to be transmitted (on the other simplex connection) before sending more data. (See qualifications, below.) Note that, due to the assynchrony of the two simplex connections, phase problems can occur with this option.
255	Command sender suggests that the other party alone should handle output linefeeds and suggests nothing about how it should be done.

The guiding rules are that:

1) if neither data receiver nor data sender wants to handle output linefeeds, the data receiver must do it, and

2) if both data receiver and data sender want to handle output linefeed disposition, the data sender gets to do it.

The reasoning for the former rule is that if neither wants to do it, then the default in the NAOLFD option dominates. If both want to do it, the sender, who is presumed to have special knowledge about the data, should be allowed to do it, taking into account any suggestions the receiver may make. Simulation is defined as the replacement of the linefeed character by new-line (see following) and enough blanks to move the print head (or line pointer) to the same lateral position it occupied just prior to receiving the linefeed. To avoid infinite recursion, such simulation is allowed only for linefeed characters that are not immediately preceded by carriage-returns (that is, part of a Telnet new-line combination). It is assumed that linefeed simulation will be necessary for printers that do not have a separate linefeed (like the IBM 2741); in this case, end-of-line character padding can be specified through NAOCRD. Any padding (0 < <8-bit-value> < 251) of linefeed characters is to be done for ALL linefeed characters.

NOTE: Delays, controlled by the data sender, must consist of NUL characters inserted immediately after the character. This is necessary due to the assynchrony of network

transmissions. Additionally, due to the presence of the Telnet end-of-line convention, it may be necessary to add carriage-return padding or delay after the associated linefeed (see NAOCRD Telnet option). As with all option negotiations, neither party should suggest a state already in effect except to refuse to negotiate; changes should be acknowledged; and once refused, an option should not be resuggested until "something changes" (e.g., another process starts). At any time, either party can disable further negotiation by giving the appropriate WON'T NAOLFD or DON'T NAOLFD command.

RFC-698 Telnet Extended ASCII Option

July 1975

Command Name and Code.

EXTEND-ASCII 17

Default

DON'T EXTEND-ASCII
WON'T EXTEND-ASCII

i.e., only use standard NVT ASCII

Command Meanings

IAC WILL EXTEND-ASCII

The sender of this command requests Permission to begin transmitting, or confirms that it may now begin transmitting extended ASCII, where additional 'control' bits are added to normal ASCII, which are treated sPecially by certain programs on the host computer.

IAC WON'T EXTEND-ASCII

If the connection is already being operated in extended ASCII mode, the sender of this command demands that the receiver begin transmitting data characters in standard NVT ASCII. If the connection is not already being operated in extended ASCII mode, The sender of this command refuses to begin transmitting extended ASCII.

IAC DO EXTEND-ASCII

The sender of this command requests that the receiver begin transmitting,or confirms that the receiver of this command is allowed to begin transmitting extended ASCII.

IAC DON'T EXTEND-ASCII

The sender of this command demands that the receiver of this command stop or not start transmitting data in extended ASCII mode.

IAC SB EXTASC

<high order bits (bits 15-8)><low order bits (bits 7-0)> IAC SE

This command transmits an extended ASCII character in the form of two 8-bit bytes. Each 8-bit byte contains 8 data bits.

RFC-698 Telnet Extended ASCII Option

Motivation

Several sites on the net, for example, SU-AI and MIT-AI, use keyboards which use almost all 128 characters as printable characters, and use one or more additional bits as "control' bits as command modifiers or to separate textual input from command input to programs. Without these additional bits, several characters cannot be entered as text because they are used for control purposes, such as the greek letter "beta' which on a TELNET connection is CONTROL-C and is used for stopping ones job. In addition there are several commonly used programs at these sites which require these additional bits to be run effectively. Hence it is necessary to provide some means of sending characters larger than 8 bits wide.

RFC-698 Telnet Extended ASCII Option

Description of the Option

This option is to allow the transmission of extended ASCII.

Experience has shown that most of the time, 7-bit ASCII is typed, with an occasional "control' character used. Hence, it is expected normal NVT ASCII would be used for 7-bit ASCII and that extended-ASCII be sent as an escape character sequence.

The exact meaning of these additional bits depends on the user program. At SU-AI and at MIT-AI, the first two bits beyond the normal 7-bit ASCII are passed on to the user program and are denoted as follows.

Bit 8 (or 200 octal) is the CONTROL bit

Bit 9 (or 400 octal) is the META bit

NOTE: "CONTROL' is used in a non-standard way here; that is, it usually refers to codes 0-37 in NVT ASCII. CONTROL and META are echoed by prefixing the normal character with 013 (integral symbol) for CONTROL and 014 (plus-minus) for META. If both are present, it is known as CONTROL-META and echoed as 013 014 7-bit character.

RFC-698 Telnet Extended ASCII Option

Description of Stanford Extended ASCII Characters

In this section, the extended graphic character set used at SU-AI is described for reference, although this specific character set is not required as part of the extended ASCII Telnet option. Characters described as "hidden" are alternate graphic interpretations of codes normally used as format effectors, used by certain typesetting programs.

Code	Graphic represented
000	null (hidden vertically centered dot)
001	downward arrow
002	alpha (all Greek letters are lowercase)
003	beta
004	logical and (caret)
005	logical not (dash with downward extension)
006	epsilon
007	pi
010	lambda
011	tab (hidden gamma)
012	linefeed (hidden delta)
013	vertical tab (hidden integral)
014	formfeed (hidden plus-minus)
015	carriage return (hidden circled-plus)
016	infinity
017	del (partial differential)
020	proper subset (right-opening horseshoe)
021	proper superset (left-opening horseshoe)
022	intersection (down-opening horseshoe)
023	union (up-opening horseshoe)
024	universal quantifier (upside-down A)
025	existential quantifier (backwards E)
026	circled-times
027	left-right double headed arrow
030	underbar
031	right pointing arrow
032	tilde
033	not-equal
034	less-than-or-equal
035	greater-than-or-equal
036	equivalence (column of 3 horizontal bars)
037	logical or (V shape)
040-135	as in standard ASCII
136	upward pointing arrow
137	left pointing arrow
140-174	as in standard ASCII
175	altmode (prints as lozenge)
176	right brace
177	rubout (hidden circumflex)

RFC-727 Telnet Logout Option

Mark Crispin; MIT-AI
27 October 1977

Command name and code.

LOGOUT 18

Default.

WON'T LOGOUT
DON'T LOGOUT

No forcible logging off of the server's user process.

Command Meanings

IAC WILL LOGOUT

The sender of this command REQUESTS permission to, or confirms that it will, forcibly log off the user process at its end.

IAC WON'T LOGOUT

The sender of this command REFUSES to forcibly log off the user process at its end.

IAC DO LOGOUT

The sender of this command REQUESTS that the receiver forcibly log off the user process at the receiver's end, or confirms that the receiver has its permission to do so.

IAC DON'T LOGOUT

The sender of this command DEMANDS that the receiver not forcibly log off the user process at the receiver's end.

RFC-727 Telnet Logout Option

Motivation for the Option

Often, a runaway user process could be hung in such a state that it cannot be interrupted by normal means. Conversely, the system itself could be bottlenecked so that response delays are intolerable. A user (human or otherwise) eventually will time out out of frustration and take the drastic means of closing the connection to free itself from the hung process. In some situations, even the simple operation of logging out can take a long time.

Some systems treat a close to mean that it should log out its user process under it. However, many hosts merely "detach" the process so that an accidental close due to a user or temporary hardware error will not cause all work done on that job to be lost; when the connection is re-established, the user may "attach" back to its process. While this protection is often valuable, if the user is giving up completely on the host, it can cause this hung job to continue to load the system.

This option allows a process to instruct the server that the user process at the server's end should be forcibly logged out instead of detached. A secondary usage of this option might be for a server to warn of impending auto-logout of its user process due to inactivity.

RFC-727 Telnet Logout Option

Description of the Option

When a user decides that it no longer wants its process on the server host and decides that it does not want to wait until the host's normal log out protocol has been gone through, it sends IAC DO LOGOUT. The receiver of the command may respond with IAC WILL LOGOUT, in which case it will then forcibly log off the user process at its end. If it responds with IAC WON'T LOGOUT, then it indicates that it has not logged off the user process at its end, and if the connection is broken, the process very possibly will be detached.

A truly impatient user that feels that it must break away from the server immediately could even send IAC DO LOGOUT and then close. At the worst, the server would only ignore the request and detach the user process. A server that implements the LOGOUT option should know to log out the user process despite the sudden close and even an inability to confirm the LOGOUT request!

A Sample Implementation of the Option

The server implements the LOGOUT option both for accepting LOGOUT requests and for auto-logout warning.

Case 1:

The user connects to the server, and starts interacting with the server. For some reason, the user wishes to terminate interaction with the server, and is reluctant to go through the normal log out procedure, or perhaps the user is unable to go through the normal log out procedure. It does not want the process at the server any more, so it sends IAC DO LOGOUT. The server verifies the request with IAC WILL LOGOUT, and then forcibly logs off the user process (perhaps by using a system call that causes another process to be logged out). It does not have to close the connection unless the user closes or it wants to close. Neither does it wait until the user has received its confirmation--it starts the log out immediately so if the user has in the mean time closed the connection without waiting for confirmation, its logout request still is performed.

Case 2:

The user connects to the server, and after logging in, is idle for a while, long enough to approach the server's autologout time. The server shortly before the autologout sends IAC WILL LOGOUT; the user sees this and sends IAC DON'T LOGOUT, and continues work on the host. Nothing prevents the server from logging out the user process if inactivity continues; this can be used to prevent a malicious user from locking up a process on the server host by the simple expedient of sending IAC DON'T LOGOUT every time it sees IAC WILL LOGOUT but doing nothing else.

RFC-735 Telnet Byte Macro Option

David H. Crocker & Richard H. Gumpertz
3 November 1977

Command name and code:

BM 19

Default:

WON'T BM -- DON'T BM

No reinterpretation of data bytes is done.

RFC-735 Telnet Byte Macro Option

Command Meanings:

IAC WILL BM

The sender of this command REQUESTS or AGREES to use the BM option, and will send single data characters which are to be interpreted as if replacement data strings had been sent.

IAC WON'T BM

The sender of this option REFUSES to send single data characters which are to be interpreted as if replacement data strings had been sent. Any existing BM <macro byte> definitions are discarded (i.e., reset to their original data interpretations).

IAC DO BM

The sender REQUESTS or AGREES to have the other side (sender of WILL BM) send single data characters which are to be interpreted as if replacement data strings had been sent.

IAC DON'T BM

The sender REFUSES to allow the other side to send single data characters which are to be interpreted as if replacement data strings had been sent. Any existing BM <macro byte> definitions are to be discarded.

IAC SB BM <DEFINE> <macro byte> <count>
<replacement string> IAC SE

where:

<macro byte> is the data byte actually to be sent across the network; it may NOT be Telnet IAC (decimal 255, but may be any other 8-bit character.

<count> is one 8-bit byte binary number, indicating how many <replacement string> characters follow, up to the ending IAC SE, but not including it. Note that doubled IACs in the definition should only be counted as one character per pair.

<replacement string> is a string of zero or more Telnet ASCII characters and/or commands, which the <macro byte> is to represent; any character may occur within a <replacement string>. Note, however, that an IAC in the string must be doubled, to be interpreted later as an IAC; to be interpreted later as data byte 255, it must be quadrupled in the original <replacement string> specification.

The indicated <macro byte> will be sent instead of the indicated <replacement string>. The receiver of the <macro byte> (the sender of the DO BM) is to behave EXACTLY as if the <replacement string> string of bytes had instead been received from the network. This interpretation is to occur before any other Telnet interpretations, unless the <macro byte> occurs as part of a Telnet command; in this case no special interpretation is to be made. In particular, an entire Telnet subnegotiation (i.e. from IAC SB through IAC SE) is to be considered a Telnet command in which NO replacement should be done.

The effect of a particular <macro byte> may be negated by reseting it to "expand" into itself.

IAC SB BM <DEFINE> X <0> IAC SE may be used to cause X to be ignored in the data stream.

<DEFINE> is decimal 1.

IAC SB BM <ACCEPT> <macro byte> IAC SE

The receiver of the <DEFINE> for <macro byte> accepts the requested definition and will perform the indicated replacement whenever a <macro byte> is received and is not part of any IAC Telnet command sequence.

<ACCEPT> is decimal 2.

IAC SB BM <REFUSE> <macro byte> <REASON> IAC SE

The receiver of the <DEFINE> for <macro byte> refuses to perform the indicated translation from <macro byte> to <replacement string> because the particular <macro byte> is not an acceptable choice, the length of the <replacement string> exceeds available storage, the length of the actual <replacement string> did not match the length predicted in the <count>, or for some unspecified reason.

<REFUSE> is decimal 3.

<REASON> may be:

<Bad-Choice> which is decimal 1;

<Too-Long> (for receiver's storage) which is decimal 2;

<Wrong-Length> (of actual string compared with promised length in <count>) which is decimal 3; or

<Other-Reason> (intended for use only until this document can be updated to include reasons not anticipated by the authors) which is decimal zero (0).

IAC SB BM <LITERAL> <macro byte> IAC SE

The <macro byte> is to be treated as real data, rather than as representative of the <replacement string>

Note that this subcommand cannot be used during Telnet subcommands, since subcommands are defined to end with the next occurrence of "IAC SE". Including this BM subcommand within any Telnet subcommand would therefore prematurely terminate the containing subcommand.

<LITERAL> is decimal 4.

IAC SB BM <PLEASE CANCEL> <macro byte> <REASON> IAC SE

The RECEIVER of the defined <macro byte> (i.e., the sender of IAC DO BM) requests the sender of <macro byte> to cancel its definition. <REASON> is the same as for the <REFUSE> subcommand.

The <macro byte> sender should (but is not required to) respond by resetting <macro byte> (i.e., sending an IAC SB BM <DEFINE> <macro byte> <1> <macro byte> IAC SE).

If the receiver absolutely insists on cancelling a given macro, the best it can do is to turn off the entire option, with IAC DONT BM, wait for an acknowledging IAC WONT BM and then restart the option, with IAC DO BM. This will reset all other macroes as well but it will allow the receiver to REFUSE with code BAD CHOICE if/when the foreign site attempts to redefine the macro in question.

RFC-735 Telnet Byte Macro Option

Motivation for the Option

Subcommands for Telnet options currently require a minimum of five characters to be sent over the network (i.e., IAC SB <Option name> IAC SE). For subcommands which are employed infrequently, in absolute numbers and in relation to normal data, this overhead is tolerable. In other cases, however, it is not. For example, data which is sent in a block-oriented fashion may need a "block separator" mark. If blocks are commonly as small as five or ten bytes, then most of the cross-net data will be control information. The BM option is intended as a simple data compression technique, to remove this overhead from the communication channel.

RFC-735 Telnet Byte Macro Option

Description of the Option

The option is enabled through the standard Telnet Option negotiation process. Afterwards, the SENDER of data (the side which sends the IAC WILL BM) is free to define and use mappings between single and replacement NVT characters. Except for the ability to refuse particular definitions, the receiver of data has no control over the definition and use of mappings.

The sender (of the WILL BM) is prohibited from using or redefining a <macro byte> until it has received an <ACCEPT> <REFUSE>, or DONT BM, in reply to a <DEFINE>.

NOTE: The Telnet command character IAC (decimal 255) may be a member of a <replacement string> but is the ONLY character which may NOT be defined as a <macro byte>.

Within any Telnet command (i.e., any sequence beginning with IAC) macro replacement may NOT take place. Data are to be interpreted only as their normal character values. This avoids the problem of distinguishing between a character which is to be taken as a <macro byte>, and interpreted as its corresponding <replacement string>, and one which is to be taken as its usual Telnet NVT value. In all other cases, however, <macro byte>s are to be interpreted immediately, as if their corresponding <replacement string>s had actually been sent across the network. Expanded strings are not subject to reinterpretation, so that recursive definitions cannot be made. Telnet commands may be included in <replacement strings>; however, they must be totally contained within the macro or must begin within the macro and terminate outside of it. In particular, they may NOT begin outside a macro and continue or terminate inside one, since no macro replacement takes place while processing any Telnet command.

Note that when skipping data due to Telnet SYNCH (INS/DM) processing, BM macro replacement should still take place, since (for example) "IAC DM" would be a valid <replacement string>.

The <count> in the <DEFINE> subcommand is intended to allow the receiver to allocate storage. IAC interpretation is not over-ridden during BM subcommands so that IAC SE will continue to safely terminate malformed subcommands.

The BM option is notably inefficient with regard to problems during <macro byte> definition and use of <macro byte>s as real data. It is expected that relatively few <macro byte>s will be defined and that they will represent relatively short strings. Since the Telnet data space between decimal 128 and decimal 254 is not normally used, except by implementations employing the original (obsolete) Telnet protocol, it is recommended that <macro byte>s normally be drawn from that pool.

RFC-736 Telnet SUPDUP Option

Mark Crispin
Stanford University
31 October 1977

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet are expected to adopt and implement this standard.

Command Name and Code

SUPDUP 21

Default

WON'T SUPDUP
DON'T SUPDUP

The SUPDUP display protocol is not in use.

Command meanings

IAC WILL SUPDUP

The sender of this command REQUESTS permission to, or confirms that it will, use the SUPDUP display protocol

IAC WON'T SUPDUP

The sender of this command REFUSES to use the SUPDUP protocol.

IAC DO SUPDUP

The sender of this command REQUESTS that the receiver use, or grants the receiver permission to use, the SUPDUP protocol.

IAC DON'T

The sender of this command DEMANDS that the receiver not use the SUPDUP protocol.

RFC-736 Telnet SUPDUP Option

Motivation for the option

Since the publication of RFC-734 "SUPDUP Protocol", I have been requested to design an option to the TELNET protocol to provide for SUPDUP service. This option allows a host to provide SUPDUP service on the normal TELNET socket (27 octal) instead of 137 (octal) which is the normal SUPDUP ICP socket.

RFC-736 Telnet SUPDUP Option

Description of the option

A user TELNET program which wishes to use the SUPDUP display protocol instead of the NVT terminal service should send an IAC DO SUPDUP. If the server is willing to use the SUPDUP display protocol, it should respond with IAC WILL SUPDUP; otherwise it should refuse with IAC WONT SUPDUP.

For hosts which normally provide SUPDUP terminal services, the server can send IAC WILL SUPDUP upon ICP which the user may then accept or refuse.

If the SUPDUP option is in effect, no further TELNET negotiations are allowed. They are meaningless, since SUPDUP has its own facilities to perform the functions that are needed. Hence, octal 377 will become an ordinary transmitted character (in this case an invalid %TD code) instead of an IAC.

Following the mutual acceptance of the SUPDUP option, the SUPDUP negotiation proceeds as described in RFC-734 "SUPDUP Protocol".

RFC-749 Telnet SUPDUP-Output Option

Bernard Greeberg
MIT-Multics
18 September 1978

Command name and code.

SUPDUP-OUTPUT 22

Default.

WON'T SUPDUP-OUTPUT
DON'T SUPDUP-OUTPUT

i.e., the SUPDUP-OUTPUT format messages may not be transmitted.

Command meanings.

IAC WILL SUPDUP-OUTPUT

The sender of this command REQUESTS permission to transmit SUPDUP-OUTPUT format messages over the TELNET connection.

IAC WON'T SUPDUP-OUTPUT

The sender of this command STATES that he will no longer send SUPDUP-OUTPUT format messages over the TELNET connection.

IAC DO SUPDUP-OUTPUT

The sender of this command grants the receiver permission to send SUPDUP-OUTPUT format messages over the TELNET connection.

IAC DON'T SUPDUP-OUTPUT

The sender of this command DEMANDS that the receiver not send SUPDUP-OUTPUT format messages over the TELNET connection.

IAC SB SUPDUP-OUTPUT 1 <terminal-parameters> IAC SE

The sender of this command (which must be the TELNET user process) is supplying information describing the capabilities of the user process' terminal.

IAC SB SUPDUP-OUTPUT 2 n TD1 TD2 .. TDn SCx SCy IAC SE

The sender of this command, which must be the TELNET server process, is sending explicit screen control information to be carried out by the user TELNET process.

RFC-749 Telnet SUPDUP-Output Option

Motivation for the option.

The SUPDUP-OUTPUT protocol provides a means to access the virtual display support provided by the SUPDUP protocol (see RFC 734) within the context of a standard TELNET connection. This allows occasional display-oriented programs at non-display-oriented servers to take advantage of the standardized display support provided by SUPDUP. This cannot be done with the standard SUPDUP protocol or the TELNET SUPDUP option (RFC 736), for they both require that all communication after the negotiation to use SUPDUP has been completed proceed according to the protocol of RFC 734. This places upon the server total responsibility for screen management for the duration of the connection, which, by hypothesis, the non-display oriented server is not willing to accept.

User TELNET programs at display-oriented user hosts provide local screen management by mapping the NVT commands of TELNET into local screen management commands; often, this involves scrolling, end-of-page processing, line clearing etc. The SUPDUP-OUTPUT option allows a display-oriented application program at the server side to take over screen management explicitly, via the SUPDUP display control repertoire. TELNET remains in effect throughout. The IAC IP and other TELNET commands are still valid.

By means of the SUPDUP-OUTPUT option, display-oriented programs can run on the server host, and control the user host's screen explicitly. The user TELNET process sends a description of the user terminal (as specified in RFC 734) to the server TELNET process as a subnegiotiation block when the SUPDUP-OUTPUT negotiation has been successfully completed. The server TELNET process sends explicit screen control commands via subnegotiation blocks to the user TELNET process.

RFC-749 Telnet SUPDUP-Output Option

Description of the option.

The SUPDUP-OUTPUT protocol may only be initiated by the server TELNET process. A server TELNET process wishing to take advantage of the SUPDUP-OUTPUT protocol will initiate a negotiation for it by sending IAC WILL SUPDUP-OUTPUT. The user TELNET process must accept or refuse the offer by sending IAC DO SUPDUP-OUTPUT or IAC DON'T SUPDUP-OUTPUT.

If the user TELNET process agrees to support the SUPDUP-OUTPUT option, it must follow the sending of IAC DO SUPDUP-OUTPUT immediately with a description of the user's terminal. This information is described in RFC 734 as the "terminal parameters." It is to be sent as a series of six-bit bytes, one byte per eight-bit

TELNET data byte. These words may or may not contain the optional line speed and graphics capabilities parameters described by RFC 747; the first six bytes specify the count of 36-bit words to follow as described by RFC 734.

The terminal parameter block will be sent as a subnegotiation of the SUPDUP-OUTPUT option:

IAC SB SUPDUP-OUTPUT 1 byte1 byte2 ... byten IAC SE

The byte of "1" is a command code, for compatibility with future extensions. Upon receipt of the terminal parameter block from the user TELNET process, the server TELNET process may send SUPDUP-OUTPUT blocks as described below.

The server TELNET process can specify explicit control of the user host's screen by the sending of subnegotiation blocks of the SUPDUP-OUTPUT option. The format of such a block, as seen in eight-bit TELNET data bytes, is:

IAC SB SUPDUP-OUTPUT 2 N TD1 TD2 TD3 ... TDn SCx SCy IAC SE

The byte of "2" is a command code, for compatibility with future extensions. The TDm bytes are the "%TDCODEs" and printing characters of SUPDUP output of RFC 734. N is a byte containing a count of the number of TDm's in this transmission. N may be zero, and may not be greater than 254 (decimal). SCx and SCy are two bytes specifying the anticipated horizontal and vertical (respectively) coordinates of the cursor of the user host's screen after the latter has interpreted all the %TDCODEs in this transmission.

The motivation for the SCx SCy screen position specification is to allow hosts running the ITS operating system, which will transmit the TDCODEs directly into the local output system, to assert the "main program level" screen position without any interpretation of the transmitted TDCODE sequence by the user TELNET program.

The user TELNET process must manage the position of the local cursor with respect to standard TELNET NVT commands and output, and SUPDUP OUTPUT transmissions. The user TELNET process may assume that the server TELNET process is managing both NVT and SUPDUP-OUTPUT output in an integrated way.

The SUPDUP-OUTPUT option makes no statement about how input is sent; this may be negotiated via other options. By default, NVT input will be used. The user-to-server screen management commands of RFC 734 are NOT implicitly handled by IAC WILL SUPDUP-OUTPUT.

In the absence of the transmission of SUPDUP-OUTPUT subnegotiation blocks, a TELNET connection operating with the SUPDUP-OUTPUT option in effect is indistinguishable from a normal TELNET connection. Thus IAC WON'T SUPDUP-OUTPUT is highly optional, and if received by the user TELNET process, should only be used to cause a diagnostic if

SUPDUP-OUTPUT subnegotiation blocks are subsequently received. If received, the user TELNET process should respond with IAC DON'T SUPDUP OUTPUT.

Because of the optional nature of IAC WON'T SUPDUP-OUTPUT, the user TELNET process should be prepared to send the terminal parameter subnegotiation block each time IAC WILL SUPDUP-OUTPUT is received, i.e., even if the user TELNET process believes SUPDUP-OUTPUT to be in effect.

The %TDORS (output reset) code may not be sent in a SUPDUP-OUTPUT transmission. The user TELNET program may assume that no byte in a subnegotiation block will be 255 (decimal).

No multi-byte TDCODE sequence (e.g., %TDMOV, %TDILP) may be split across SUPDUP-OUTPUT subnegotiation blocks.

References:

Crispin, Mark:

"SUPDUP Display Protocol", RFC 734, 7 October 1977, NIC 44213.

Crispin, Mark:

"TELNET SUPDUP Option", RFC 736, 31 October 1977, NIC 44213.

Crispin, Mark:

"Recent Extensions to the SUPDUP Protocol", RFC 747, 21 March 1978, NIC 44015.

RFC-768 User Datagram Protocol

RFC-768 User Datagram Protocol

Introduction

This User Datagram Protocol (UDP) is defined to make available a datagram mode of packet-switched computer communication in the environment of an interconnected set of computer networks. This protocol assumes that the Internet Protocol (IP) is used as the underlying protocol.

This protocol provides a procedure for application programs to send messages to other programs with a minimum of protocol mechanism. The protocol is transaction oriented, and delivery and duplicate protection are not guaranteed. Applications requiring ordered reliable delivery of streams of data should use the Transmission Control Protocol (TCP).

Protocol Application

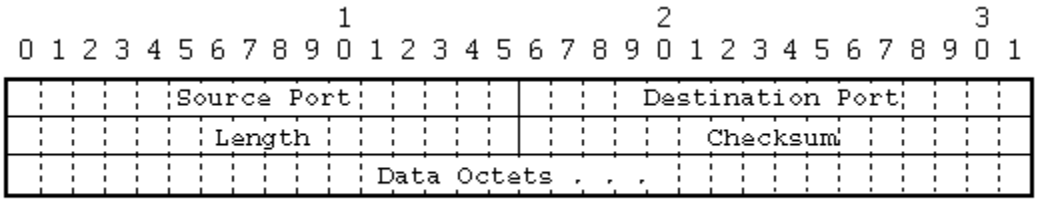
The major uses of this protocol is the Internet Name Server [3], and the Trivial File Transfer Protocol.

Protocol Number

This is protocol 17 (21 octal) when used in the Internet Protocol. Other protocol numbers are listed in Assigned Numbers.

RFC-768 User Datagram Protocol

Format



User Datagram Header Format

RFC-768 User Datagram Protocol

Fields

Source Port
An optional field, when meaningful, it indicates the port of the sending process, and may be assumed to be the port to which a reply should be addressed in the absence of any other information. If not used, a value of zero is inserted.

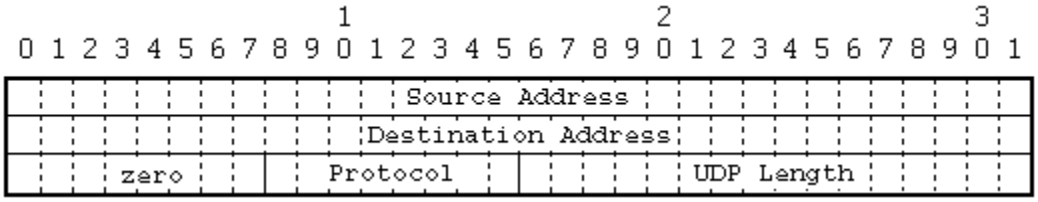
Destination Port
Has a meaning within the context of a particular internet destination address.

Length
The length in octets of this user datagram including this header and the data. (This means the minimum value of the length is eight.)

Checksum
The 16-bit one's complement of the one's complement sum of a pseudo header of information from the IP header, the UDP header, and the data, padded with zero octets at the end (if necessary) to make a multiple of two octets.

> If the computed checksum is zero, it is transmitted as all ones (the equivalent in one's complement arithmetic). An all zero transmitted checksum value means that the transmitter generated no checksum (for debugging or for higher level protocols that don't care).

The pseudo header conceptually prefixed to the UDP header contains the source address, the destination address, the protocol, and the UDP length. This information gives protection against misrouted datagrams. This checksum procedure is the same as is used in TCP.



RFC-768 User Datagram Protocol - Fields

Ports

UDP well-known ports follow the same rules as TCP well-known ports.

If a datagram arrives addressed to a UDP port for which there is no pending LISTEN call, UDP **should** send an ICMP Port Unreachable message.

Checksum

A host **must** implement the facility to generate and validate UDP checksums. An application **may** optionally be able to control whether a UDP checksum will be generated, but it **must** default to checksumming on.

If a UDP datagram is received with a checksum that is non- zero and invalid, UDP **must** silently discard the datagram. An application **may** optionally be able to control whether UDP datagrams without checksums should be discarded or passed to the application.

Discussion
Some applications that normally run only across local area networks have chosen to turn off UDP checksums for efficiency. As a result, numerous cases of undetected errors have been reported. The advisability of ever turning off UDP checksumming is very controversial.

Implementation
There is a common implementation error in UDP checksums. Unlike the TCP checksum, the UDP checksum is optional; the value zero is transmitted in the checksum field of a UDP header to indicate the absence of a checksum. If the transmitter really calculates a UDP checksum of zero, it must transmit the checksum as all 1's (65535). No special action is required at the receiver, since zero and 65535 are equivalent in 1's complement arithmetic.

User Interface

A user interface should allow

> the creation of new receive ports,
>
> receive operations on the receive ports that return the data octets and an indication of source port and source address,
>
> and an operation that allows a datagram to be sent, specifying the data, source and destination ports and addresses to be sent.

IP Interface

The UDP module must be able to determine the source and destination internet addresses and the protocol field from the internet header. One possible UDP/IP interface would return the whole internet datagram including all of the internet header in response to a receive operation. Such an interface would also allow the UDP to pass a full internet datagram complete with header to the IP to send. The IP would verify certain fields for consistency and compute the internet header checksum.

Invalid Addresses

A UDP datagram received with an invalid IP source address (e.g., a broadcast or multicast address) must be discarded by UDP or by the IP layer.

When a host sends a UDP datagram, the source address **must** be (one of) the IP address(es) of the host.

IP Options

UDP **must** pass any IP option that it receives from the IP layer transparently to the application layer.

An application **must** be able to specify IP options to be sent in its UDP datagrams, and UDP **must** pass these options to the IP layer.

> **Discussion**
> At present, the only options that need be passed through UDP are Source Route, Record Route, and Time Stamp. However, new options may be defined in the future, and UDP need not and should not make any assumptions about the format or content of options it passes to or from the application; an exception to this might be an IP-layer security option.
>
> An application based on UDP will need to obtain a source route from a request datagram and supply a reversed route for sending the corresponding reply.

ICMP Messages

UDP **must** pass to the application layer all ICMP error messages that it receives from the IP layer. Conceptually at least, this may be accomplished with an upcall to the ERROR_REPORT routine (See Asynchronous Reports).

Discussion

Note that ICMP error messages resulting from sending a UDP datagram are received asynchronously. A UDP-based application that wants to receive ICMP error messages is responsible for maintaining the state necessary to demultiplex these messages when they arrive; for example, the application may keep a pending receive operation for this purpose. The application is also responsible to avoid confusion from a delayed ICMP error message resulting from an earlier use of the same port(s).

Multihoming

When a UDP datagram is received, its specific-destination address **must** be passed up to the application layer.

An application program **must** be able to specify the IP source address to be used for sending a UDP datagram or to leave it unspecified (in which case the networking software will choose an appropriate source address). There **should** be a way to communicate the chosen source address up to the application layer (e.g, so that the application can later receive a reply datagram only from the corresponding interface).

Discussion

A request/response application that uses UDP should use a source address for the response that is the same as the specific destination address of the request. See the "General Issues" section of [RFC-1123].

UDP/Application Layer Interface

The application interface to UDP **must** provide the full services of the Internet/Transport Interface. Thus, an application using UDP needs the functions of the GET_SRCADDR(), GET_MAXSIZES(), ADVISE_DELIVPROB(), and RECV_ICMP() calls described in Internet/Transport Layer Interface. For example, GET_MAXSIZES() can be used to learn the effective maximum UDP maximum datagram size for a particular {interface,remote host,TOS} triplet.

An application-layer program **must** be able to set the TTL and TOS values as well as IP options for sending a UDP datagram, and these values must be passed transparently to the IP layer. UDP **may** pass the received TOS up to the application layer.

RFC-779 Telnet Send-Location Option

E. Killian; LLL
April 1981

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet are expected to adopt and implement this standard.

Command Name and Code

SEND-LOCATION 23

Default

WON'T SEND-LOCATION
DON'T SEND-LOCATION

Command meanings

IAC WILL SEND-LOCATION

The sender REQUESTS or AGREES to use the SEND-LOCATION option to send the user's location.

IAC WON'T SEND-LOCATION

The sender REFUSES to use the SEND-LOCATION option.

IAC DO SEND-LOCATION

The sender REQUESTS that, or AGREES to have, the other side use SEND-LOCATION commands send the user's location.

IAC DON'T SEND-LOCATION

The sender DEMANDS the other side not use the SEND-LOCATION option.

IAC SB SEND-LOCATION <location> IAC SE

The sender specifies the user's location to the other side via a SEND-LOCATION subnegotiation. <location> is a sequence of ASCII printable characters; it is terminated by the IAC SE.

RFC-779 Telnet Send-Location Option

Motivation for the option

Many network sites now provide a listing of the users currently logged in giving their names and locations (see the NAME/FINGER protocol, RFC 742). The location is useful for physically locating the user if he or she is nearby, or for calling them (a nearby phone number is often included). However, for users logged in via the network, the location printed is often no more than the originating site name. This TELNET option allows the user's TELNET program to send the user's location to the server TELNET so that it can be displayed in addition to the site name. This functionality is already present in the SUPDUP protocol (RFC 734).

RFC-779 Telnet Send-Location Option

Description of the option

When the user TELNET program knows the user's location, it should offer to transmit this information to the server TELNET by sending IAC WILL SEND-LOCATION. If the server's system is able to make use of this information (as can the ITS sites), then the server will reply with IAC DO SEND-LOCATION. The user TELNET is then free to send the location in a subnegotiation at any time.

RFC-783 The TFTP Protocol (Rev 2)

K. R. Sollins
MIT
June 1981

RFC-783 Trivial File Transfer Protocol (TFTP)

Summary

TFTP is a very simple protocol used to transfer files. It is from this that its name comes, Trivial File Transfer Protocol or TFTP. Each nonterminal packet is acknowledged separately. This document describes the protocol and its types of packets. The document also explains the reasons behind some of the design decisions.

Acknowledgements

The protocol was originally designed by Noel Chiappa, and was redesigned by him, Bob Baldwin and Dave Clark, with comments from Steve Szymanski. The current revision of the document includes modifications stemming from discussions with and suggestions from Larry Allen, Noel Chiappa, Dave Clark, Geoff Cooper, Mike Greenwald, Liza Martin, David Reed, Craig Milo Rogers (of UCS-ISI), Kathy Yellick, and the author.

The acknowledgement and retransmission scheme was inspired by TCP, and the error mechanism was suggested by PARC's EFTP abort message.

RFC-783 Trivial File Transfer Protocol (TFTP)

Purpose

TFTP is a simple protocol to transfer files, and therefore was named the Trivial File Transfer Protocol or TFTP. It has been implemented on top of the Internet User Datagram protocol (UDP or Datagram) so it may be used to move files between machines on different networks implementing UDP. (This should not exlude the possibility of implementing TFTP on top of other datagram protocols.) It is designed to be small and easy to implement. Therefore, it lacks most of the features of a regular FTP. The only thing it can do is read and write files (or mail) from/to a remote server. It cannot list directories, and currently has no provisions for user authentication. In common with other Internet protocols, it passes 8 bit bytes of data.

Three modes of transfer are currently supported: netascii ; octet , raw 8 bit bytes; mail, netascii characters sent to a user rather than a file. Additional modes can be defined by pairs of cooperating hosts.

NOTE: The transfer mode "MAIL" should not be supported.

RFC-783 Trivial File Transfer Protocol (TFTP)

Overview of the Protocol

Any transsfer begins with a request to read or write a file, which also serves to request a connection. If the server grants the request, the connection is opened and the file is sent in fixed length blocks of 512 bytes. Each data packet contains one block of data, and must be acknowledged by an acknowledgment packet before the next packet can be sent. A data packet of less than 512 bytes signals termination of a transfer. If a packet gets lost in the network, the intended recipient will timeout and may retransmit his last packet (which may be data or an acknowledgment), thus causing the sender of the lost packet to retransmit that lost packet. The sender has to keep just one packet on hand for retransmission, since the lock step acknowledgment guarantees that all older packets have been received. Notice that both machines involved in a transfer are considered senders and receivers. One sends data and receives acknowledgments, the other sends acknowledgments and receives data.

Most errors cause termination of the connection. An error is signalled by sending an error packet. This packet is not acknowledged, and not retransmitted (i.e., a TFTP server or user may terminate after sending an error message), so the other end of the connection may not get it. Therefore timeouts are used to detect such a termination when the error packet has been lost. Errors are caused by three types of events: not being able to satisfy the request (e.g., file not found, access violation, or no such user), receiving a packet which cannot be explained by a delay or duplication in the network (e.g. an incorrectly formed packet), and losing access to a necessary resource (e.g., disk full or access denied during a transfer).

TFTP recognizes only one error condition that does not cause termination, the source port of a received packet being incorrect. In this case, an error packet is sent to the originating host.

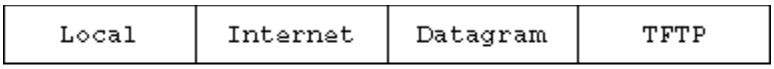
This protocol is very restrictive, in order to simplify implementation. For example, the fixed length blocks make allocation straight forward, and the lock step acknowledgement provides flow control and eliminates the need to reorder incoming data packets.

RFC-783 Trivial File Transfer Protocol (TFTP)

Relation to other Protocols

As mentioned TFTP is designed to be implemented on top of the Datagram protocol. Since Datagram is implemented on the Internet protocol, packets will have an Internet header, a Datagram header, and a TFTP header. Additionally, the packets may have a header (LNI, ARPA header, etc.) to allow them through the local transport medium. As shown in Figure 3-1, the order of the contents of a packet will be: local medium header, if used, Internet header, Datagram header, TFTP header, followed by the remainder of the TFTP packet. (This may or may not be data depending on the type of packet as specified in the TFTP header.) TFTP does not specify any of the values in the Internet header. On the other hand, the source and destination port fields of the Datagram header (its format is given in the appendix) are used by TFTP and the length field reflects the size of the TFTP packet. The transfer identifiers (TID's) used by TFTP are passed to the Datagram layer to be used as ports; therefore they must be between 0 and 65,535. The initialization of TID's is discussed in the section on initial connection protocol.

The TFTP header consists of a 2 byte opcode field which indicates the packet's type (e.g., DATA, ERROR, etc.) These opcodes and the formats of the various types of packets are discussed further in the section on TFTP packets.



Order of Headers

Initial Connection Protocol

A transfer is established by sending a request (WRQ to write onto a foreign file system, or RRQ to read from it), and receiving a positive reply, an acknowledgment packet for write, or the first data packet for read. In general an acknowledgment packet will contain the block number of the data packet being acknowledged. Each data packet has associated with it a block number; block numbers are consecutive and begin with one. Since the positive response to a write request is an acknowledgment packet, in this special case the block number will be zero. (Normally, since an acknowledgment packet is acknowledging a data packet, the acknowledgment packet will contain the block number of the data packet being acknowledged.) If the reply is an error packet, then the request has been denied.

In order to create a connection, each end of the connection chooses a TID for itself, to be used for the duration of that connection. The TID's chosen for a connection should be randomly chosen, so that the probability that the same number is chosen twice in immediate succession is very low. Every packet has associated with it the two TID's of the ends of the connection, the source TID and the destination TID. These TID's are handed to the supporting UDP (or other datagram protocol) as the source and destination ports. A requesting host chooses its source TID as described above, and sends its initial request to the known TID 69 decimal (105 octal) on the serving host. The response to the request, under normal operation, uses a TID chosen by the server as its source TID and the TID chosen for the previous message by the requestor as its destination TID. The two chosen TID's are then used for the remainder of the transfer.

As an example, the following shows the steps used to establish a connection to write a file. Note that WRQ, ACK, and DATA are the names of the write request, acknowledgment, and data types of packets respectively. The appendix contains a similar example for reading a file.

1. Host A sends a "WRQ" to host B with source= A's TID, destination= 69.
2. Host B sends a "ACK" (with block number= 0) to host A with source= B's TID, destination= A's TID.

At this point the connection has been established and the first data packet can be sent by Host A with a sequence number of 1. In the next step, and in all succeeding steps, the hosts should make sure that the source TID matches the value that was agreed on in steps 1 and 2. If a source TID does not match, the packet should be discarded as erroneously sent from somewhere else. An error packet should be sent to the source of the incorrect packet, while not disturbing the transfer.

This can be done only if the TFTP in fact receives a packet with an incorrect TID. If the supporting protocols do not allow it, this particular error condition will not arise.

The following example demonstrates a correct operation of the protocol in which the above situation can occur. Host A sends a request to host B. Somewhere in the network, the request packet is duplicated, and as a result two acknowledgments are returned to host A, with different TID's chosen on host B in response to the two requests. When the first response arrives, host A continues the connection. When the second response to the request arrives, it should be rejected, but there is no reason to terminate the first connection. Therefore, if different TID's are chosen for the two connections on host B and host A checks the source TID's of the messages it receives, the first connection can

be maintained while the second is rejected by returning an error packet.

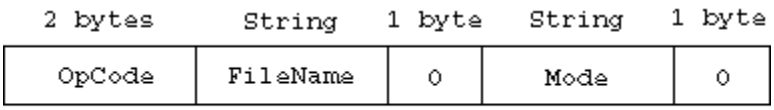
RFC-783 Trivial File Transfer Protocol (TFTP)

TFTP Packets

TFTP supports five types of packets, all of which have been mentioned above:

opcode	operation
1	Read request (RRQ)
2	Write request (WRQ)
3	Data (DATA)
4	Acknowledgment (ACK)
5	Error (ERROR)

The TFTP header of a packet contains the opcode associated with that packet.

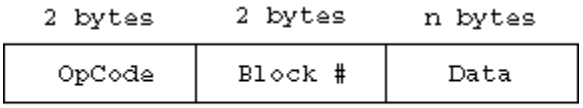


RRQ/WRQ Packet

RRQ and WRQ packets (opcodes 1 and 2 respectively) have the format shown in the figure above. The file name is a sequence of bytes in netascii terminated by a zero byte. The mode field contains the string "netascii", "octet", or "mail" (or any comibnation of upper and lower case, such as "NETASCII", NetAscii", etc.) in netascii indicating the three modes defined in the protocol. A host which receives netascii mode data must translate the data to its own format. Octet mode is used to transfer a file that is in the 8-bit format of the machine from which the file is being transferred. It is assumed that each type of machine has a single 8-bit format that is more common, and that that format is chosen. For example, on a DEC-20, a 36 bit machine, this is four 8-bit bytes to a word with four bits of breakage. If a host receives a octet file and then returns it, the returned file must be identical to the original. Mail mode uses the name of a mail recipient in place of a file and must begin with a WRQ. Otherwise it is identical to netascii mode. The mail recipient string should be of the form "username" or "username@hostname". If the second form is used, it allows the option of mail forwarding by a relay computer.

The discussion above assumes that both the sender and recipient are operating in the same mode, but there is no reason that this has to be the case. For example, one might build a storage server. There is no reason that such a machine needs to translate netascii into its own form of text. Rather, the sender might send files in netascii, but the storage server might simply store them without translation in 8-bit format. Another such situation is a problem that currently exists on DEC-20 systems. Neither netascii nor octet accesses all the bits in a word. One might create a special mode for such a machine which read all the bits in a word, but in which the receiver stored the information in 8-bit format. When such a file is retrieved from the storage site, it must be restored to its original form to be useful, so the reverse mode must also be implemented. The user site will have to remember some information to achieve this. In both of these examples, the request packets would specify octet mode to the foreign host, but the local host would be in some other mode. No such machine or application specific modes have been specified in TFTP, but one would be compatible with this specification.

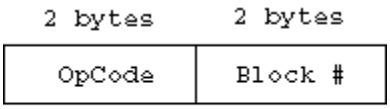
It is also possible to define other modes for cooperating pairs of hosts, although this must be done with care. There is no requirement that any other hosts implement these. There is no central authority that will define these modes or assign them names.



DATA Packet

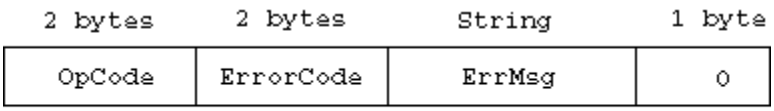
Data is actually transferred in DATA packets depicted in the above figure. DATA packets (opcode = 3) have a block number and data field. The block numbers on data packets begin with one and increase by one for each new block of data. This restriction allows the program to use a single number to discriminate between new packets and duplicates. The data field is from zero to 512 bytes long. If it is 512 bytes long, the block is not the last block of data; if it is from zero to 511 bytes long, it signals the end of the transfer. (See the section on Normal Termination for details.)

All packets other than those used for termination are acknowledged individually unless a timeout occurs. Sending a DATA packet is an acknowledgment for the ACK packet of the previous DATA packet. The WRQ and DATA packets are acknowledged by ACK or ERROR packets, while RRQ and



ACK Packet

ACK packets are acknowledged by DATA or ERROR packets. The figure above depicts an ACK packet; the opcode is 4. The block number in an ACK echoes the block number of the DATA packet being acknowledged. A WRQ is acknowledged with an ACK packet having a block number of zero.



Error Packet

An ERROR packet (opcode 5) takes the form depicted in the figure above. An ERROR packet can be the acknowledgment of any other type of packet. The error code is an integer indicating the nature of the error. A table of values and meanings is given in the appendix. (Note that several error codes have been added to this version of this document.) The error message is intended for human consumption, and should be in netascii.

Like all other strings, it is terminated with a zero byte.

Normal Termination

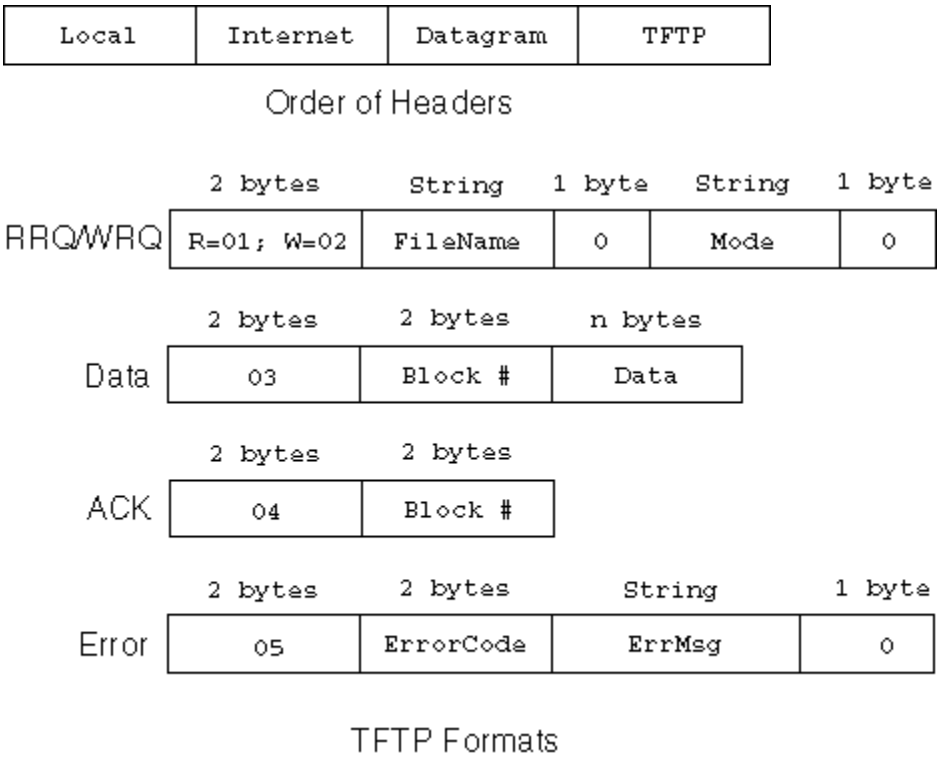
The end of a transfer is marked by a DATA packet that contains between 0 and 511 bytes of data (i.e. Datagram length < 516). This packet is acknowledged by an ACK packet like all other DATA packets. The host acknowledging the final DATA packet may terminate its side of the connection on sending the final ACK. On the other hand, dallying is encouraged. This means that the host sending the final ACK will wait for a while before terminating in order to retransmit the final ACK if it has been lost. The acknowledger will know that the ACK has been lost if it receives the final DATA packet again. The host sending the last DATA must retransmit it until the packet is acknowledged or the sending host times out. If the response is an ACK, the transmission was completed successfully. If the sender of the data times out and is not prepared to retransmit any more, the transfer may still have been completed successfully, after which the acknowledger or network may have experienced a problem. It is also possible in this case that the transfer was unsuccessful. In any case, the connection has been closed.

Premature Termination

If a request can not be granted, or some error occurs during the transfer, then an ERROR packet (opcode 5) is sent. This is only a courtesy since it will not be retransmitted or acknowledged, so it may never be received. Timeouts must also be used to detect errors.

RFC-783 Trivial File Transfer Protocol (TFTP)

Appendix

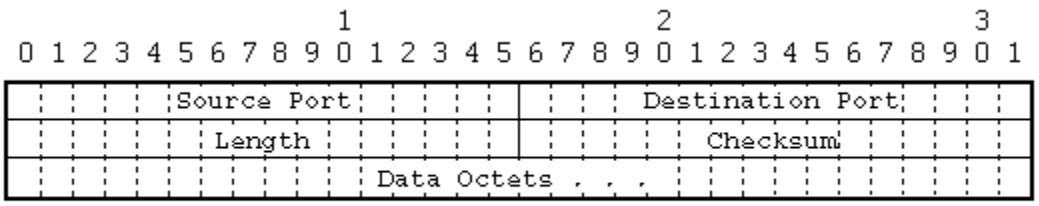


Initial Connection Protocol for reading a file

1. Host A sends a "RRQ" to host B with source= A's TID, destination= 69.
2. Host B sends a "DATA" (with block number= 1) to host A with source= B's TID, destination= A's TID.

Error Codes

Value	Meaning
0	Not defined, see error message (if any).
1	File not found.
2	Access violation.
3	Disk full or allocation exceeded.
4	Illegal TFTP operation.
5	Unknown transfer ID.
6	File already exists.
7	No such user.



User Datagram Header Format

Values of Fields

Source Port	Picked by originator of packet.
Dest. Port	Picked by destination machine (69 for RRQ or WRQ).
Length	Number of bytes in packet after Datagram header.
Checksum	Reference 2 describes rules for computing checksum. Field contains zero if unused.

Note: TFTP passes transfer identifiers (TID's) to the Internet User Datagram protocol to be used as the source and destination ports.

RFC-791 Internet Protocol

Jon Postel
USC/Information Sciences Institute
September 1981

This section includes the original text of RFC-791 as well as many corrections, comments, and suggestions made since its publication. There are also links to other relevant documents and discussions.

RFC-791 Internet Protocol

Introduction

RFC-791 Internet Protocol

Overview

RFC-791 Internet Protocol

Specification

Preface

This document specifies the DoD Standard Internet Protocol. This document is based on six earlier editions of the ARPA Internet Protocol Specification, and the present text draws heavily from them. There have been many contributors to this work both in terms of concepts and in terms of text. This edition revises aspects of addressing, error handling, option codes, and the security, precedence, compartments, and handling restriction features of the internet protocol.

Jon Postel
Editor

Motivation

The Internet Protocol is designed for use in interconnected systems of packet-switched computer communication networks. Such a system has been called a "catenet" [1]. The internet protocol provides for transmitting blocks of data called datagrams from sources to destinations, where sources and destinations are hosts identified by fixed length addresses. The internet protocol also provides for fragmentation and reassembly of long datagrams, if necessary, for transmission through "small packet" networks.

Scope

The internet protocol is specifically limited in scope to provide the functions necessary to deliver a package of bits (an internet datagram) from a source to a destination over an interconnected system of networks. There are no mechanisms to augment end-to-end data reliability, flow control, sequencing, or other services commonly found in host-to-host protocols. The internet protocol can capitalize on the services of its supporting networks to provide various types and qualities of service.

Interfaces

This protocol is called on by host-to-host protocols in an internet environment. This protocol calls on local network protocols to carry the internet datagram to the next gateway or destination host.

For example, a TCP module would call on the internet module to take a TCP segment (including the TCP header and user data) as the data portion of an internet datagram. The TCP module would provide the addresses and other parameters in the internet header to the internet module as arguments of the call. The internet module would then create an internet datagram and call on the local network interface to transmit the internet datagram.

In the ARPANET case, for example, the internet module would call on a local net module which would add the 1822 leader [2] to the internet datagram creating an ARPANET message to transmit to the IMP. The ARPANET address would be derived from the internet address by the local network interface and would be the address of some host in the ARPANET, that host might be a gateway to other networks.

Operation

The internet protocol implements two basic functions: addressing and fragmentation.

The internet modules use the addresses carried in the internet header to transmit internet datagrams toward their destinations. The selection of a path for transmission is called routing.

The internet modules use fields in the internet header to fragment and reassemble internet datagrams when necessary for transmission through "small packet" networks.

The model of operation is that an internet module resides in each host engaged in internet communication and in each gateway that interconnects networks. These modules share common rules for interpreting address fields and for fragmenting and assembling internet datagrams. In addition, these modules (especially in gateways) have procedures for making routing decisions and other functions.

The internet protocol treats each internet datagram as an independent entity unrelated to any other internet datagram. There are no connections or logical circuits (virtual or otherwise).

The internet protocol uses four key mechanisms in providing its service: Type of Service, Time to Live, Options, and Header Checksum.

The **Type of Service** is used to indicate the quality of the service desired. The type of service is an abstract or generalized set of parameters which characterize the service choices provided in the networks that make up the internet. This type of service indication is to be used by gateways to select the actual transmission parameters for a particular network, the network to be used for the next hop, or the next gateway when routing an internet datagram.

The **Time to Live** is an indication of an upper bound on the lifetime of an internet datagram. It is set by the sender of the datagram and reduced at the points along the route where it is processed. If the time to live reaches zero before the internet datagram reaches its destination, the internet datagram is destroyed. The time to live can be thought of as a self destruct time limit.

The **Options** provide for control functions needed or useful in some situations but unnecessary for the most common communications. The options include provisions for timestamps, security, and special routing.

The **Header Checksum** provides a verification that the information used in processing internet datagram has been transmitted correctly. The data may contain errors. If the header checksum fails, the internet datagram is discarded at once by the entity which detects the error.

The internet protocol does not provide a reliable communication facility. There are no acknowledgments either end-to-end or hop-by-hop. There is no error control for data, only a header checksum. There are no retransmissions. There is no flow control.

Errors detected may be reported via the Internet Control Message Protocol (ICMP) described in RFC-792 which is implemented in the internet protocol module.

Routing Outbound Datagrams

The IP layer chooses the correct next hop for each datagram it sends. If the destination is on a connected network, the datagram is sent directly to the destination host; otherwise, it has to be routed to a gateway on a connected network.

Local/Remote Decision

To decide if the destination is on a connected network, the following algorithm, described in "Internet Standard Subnetting" [RFC-950] **must** be used:

(a) The address mask (particular to a local IP address for a multihomed host) is a 32-bit mask that selects the network number and subnet number fields of the corresponding IP address.

(b) If the IP destination address bits extracted by the address mask match the IP source address bits extracted by the same mask, then the destination is on the corresponding connected network, and the datagram is to be transmitted directly to the destination host.

(c) If not, then the destination is accessible only through a gateway. Selection of a gateway is described below (Gateway Selection).

A special-case destination address is handled as follows:

* For a limited broadcast or a multicast address, simply pass the datagram to the link layer for the appropriate interface.

* For a (network or subnet) directed broadcast, the datagram can use the standard routing algorithms.

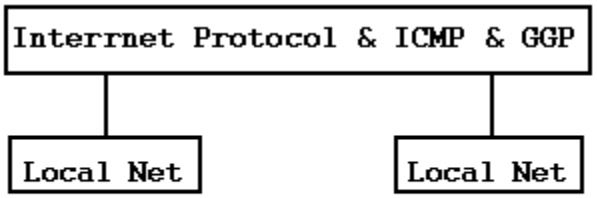
The host IP layer **must** operate correctly in a minimal network environment, and in particular, when there are no gateways. For example, if the IP layer of a host insists on finding at least one gateway to initialize, the host will be unable to operate on a single isolated broadcast net.

RFC-791 Internet Protocol - Routing

Gateways

Gateways implement internet protocol to forward datagrams between networks. Gateways also implement the Gateway to Gateway Protocol (GGP) [7] to coordinate routing and other internet control information.

In a gateway the higher level protocols need not be implemented and the GGP functions are added to the IP module.



Gateway Protocols

Gateway Selection

To efficiently route a series of datagrams to the same destination, the source host **must** keep a "route cache" of mappings to next-hop gateways. A host uses the following basic algorithm on this cache to route a datagram; this algorithm is designed to put the primary routing burden on the gateways (See RFC-816 "Fault Isolation and Recovery").

(a) If the route cache contains no information for a particular destination, the host chooses a "default" gateway and sends the datagram to it. It also builds a corresponding Route Cache entry.

(b) If that gateway is not the best next hop to the destination, the gateway will forward the datagram to the best next-hop gateway and return an ICMP Redirect message to the source host.

(c) When it receives a Redirect, the host updates the next-hop gateway in the appropriate route cache entry, so later datagrams to the same destination will go directly to the best gateway.

Since the subnet mask appropriate to the destination address is generally not known, a Network Redirect message **should** be treated identically to a Host Redirect message; i.e., the cache entry for the destination host (only) would be updated (or created, if an entry for that host did not exist) for the new gateway.

Discussion
This recommendation is to protect against gateways that erroneously send Network Redirects for a subnetted network, in violation of the gateway requirements [RFC-1009].

When there is no route cache entry for the destination host address (and the destination is not on the connected network), the IP layer **must** pick a gateway from its list of "default" gateways. The IP layer **must** support multiple default gateways.

As an extra feature, a host IP layer **may** implement a table of "static routes". Each such static route **may** include a flag specifying whether it may be overridden by ICMP Redirects.

Discussion
A host generally needs to know at least one default gateway to get started. This information can be obtained from a configuration file or else from the host startup sequence, e.g., the BOOTP Protocol [RFC-951].

It has been suggested that a host can augment its list of default gateways by recording any new gateways it learns about. For example, it can record every gateway to which it is ever redirected. Such a feature, while possibly useful in some circumstances, may cause problems in other cases (e.g., gateways are not all equal), and it is not recommended.

A static route is typically a particular preset mapping from destination host or network into a particular next-hop gateway; it might also depend on the Type-of- Service (see next section). Static routes would be set up by system administrators to override the normal automatic routing mechanism, to handle exceptional situations. However, any static routing information is a potential source of failure as configurations change or equipment fails.

Route Cache

Each route cache entry needs to include the following fields:

(1) Local IP address (for a multihomed host)

(2) Destination IP address

(3) Type(s)-of-Service

(4) Next-hop gateway IP address

Field (2) **may** be the full IP address of the destination host, or only the destination network number. Field (3), the TOS, **should** be included.

See Multihoming Requirements for a discussion of the implications of multihoming for the lookup procedure in this cache.

Discussion
Implementation

Route Cache Discussion

Including the Type-of-Service field in the route cache and considering it in the host route algorithm will provide the necessary mechanism for the future when Type-of-Service routing is commonly used in the Internet.

Each route cache entry defines the endpoints of an Internet path. Although the connecting path may change dynamically in an arbitrary way, the transmission characteristics of the path tend to remain approximately constant over a time period longer than a single typical host-host transport connection. Therefore, a route cache entry is a natural place to cache data on the properties of the path. Examples of such properties might be the maximum unfragmented datagram size (see Fragmentation), or the average round-trip delay measured by a transport protocol. This data will generally be both gathered and used by a higher layer protocol, e.g., by TCP, or by an application using UDP. Experiments are currently in progress on caching path properties in this manner.

There is no consensus on whether the route cache should be keyed on destination host addresses alone, or allow both host and network addresses. Those who favor the use of only host addresses argue that:

(1) As required in Gateway Selection, Redirect messages will generally result in entries keyed on destination host addresses; the simplest and most general scheme would be to use host addresses always.

(2) The IP layer may not always know the address mask for a network address in a complex subnetted environment.

(3) The use of only host addresses allows the destination address to be used as a pure 32-bit number, which may allow the Internet architecture to be more easily extended in the future without any change to the hosts.

The opposing view is that allowing a mixture of destination hosts and networks in the route cache:

(1) Saves memory space.

(2) Leads to a simpler data structure, easily combining the cache with the tables of default and static routes.

(3) Provides a more useful place to cache path properties, as discussed earlier.

Route Cache Implementation

The cache needs to be large enough to include entries for the maximum number of destination hosts that may be in use at one time.

A route cache entry may also include control information used to choose an entry for replacement. This might take the form of a "recently used" bit, a use count, or a last-used timestamp, for example. It is recommended that it include the time of last modification of the entry, for diagnostic purposes.

An implementation may wish to reduce the overhead of scanning the route cache for every datagram to be transmitted. This may be accomplished with a hash table to speed the lookup, or by giving a connection- oriented transport protocol a "hint" or temporary handle on the appropriate cache entry, to be passed to the IP layer with each subsequent datagram.

Although we have described the route cache, the lists of default gateways, and a table of static routes as conceptually distinct, in practice they may be combined into a single "routing table" data structure.

Dead Gateway Detection

The IP layer **must** be able to detect the failure of a "next- hop" gateway that is listed in its route cache and to choose an alternate gateway (see Identification).

Dead gateway detection is covered in some detail in RFC-816 "Fault Isolation and Recovery". Experience to date has not produced a complete algorithm which is totally satisfactory, though it has identified several forbidden paths and promising techniques.

* A particular gateway **should not** be used indefinitely in the absence of positive indications that it is functioning.
* Active probes such as "pinging" (i.e., using an ICMP Echo Request/Reply exchange) are expensive and scale poorly. In particular, hosts **must not** actively check the status of a first-hop gateway by simply pinging the gateway continuously.
* Even when it is the only effective way to verify a gateway's status, pinging **must** be used only when traffic is being sent to the gateway and when there is no other positive indication to suggest that the gateway is functioning.
* To avoid pinging, the layers above and/or below the Internet layer **should** be able to give "advice" on the status of route cache entries when either positive (gateway OK) or negative (gateway dead) information is available.

Dead Gateway Detection Discussion

If an implementation does not include an adequate mechanism for detecting a dead gateway and re-routing, a gateway failure may cause datagrams to apparently vanish into a "black hole". This failure can be extremely confusing for users and difficult for network personnel to debug.

The dead-gateway detection mechanism must not cause unacceptable load on the host, on connected networks, or on first-hop gateway(s). The exact constraints on the timeliness of dead gateway detection and on acceptable load may vary somewhat depending on the nature of the host's mission, but a host generally needs to detect a failed first-hop gateway quickly enough that transport-layer connections will not break before an alternate gateway can be selected.

Passing advice from other layers of the protocol stack complicates the interfaces between the layers, but it is the preferred approach to dead gateway detection. Advice can come from almost any part of the IP/TCP architecture, but it is expected to come primarily from the transport and link layers. Here are some possible sources for gateway advice:

- TCP or any connection-oriented transport protocol **should** be able to give negative advice, e.g., triggered by excessive retransmissions.
- TCP **may** give positive advice when (new) data is acknowledged. Even though the route may be asymmetric, an ACK for new data proves that the acknowleged data must have been transmitted successfully.
- An ICMP Redirect message from a particular gateway **should** be used as positive advice about that gateway.
- Link-layer information that reliably detects and reports host failures (e.g., ARPANET Destination Dead messages) **should** be used as negative advice.
- Failure to ARP or to re-validate ARP mappings **may** be used as negative advice for the corresponding IP address.
- Packets arriving from a particular link-layer address are evidence that the system at this address is alive. However, turning this information into advice about gateways requires mapping the link-layer address into an IP address, and then checking that IP address against the gateways pointed to by the route cache. This is probably prohibitively inefficient.

Note that positive advice that is given for every datagram received may cause unacceptable overhead in the implementation.

While advice might be passed using required arguments in all interfaces to the IP layer, some transport and application layer protocols cannot deduce the correct advice. These interfaces must therefore allow a neutral value for advice, since either always-positive or always-negative advice leads to incorrect behavior.

There is another technique for dead gateway detection that has been commonly used but is not recommended.

This technique depends upon the host passively receiving ("wiretapping") the Interior Gateway Protocol (IGP) datagrams that the gateways are broadcasting to each other. This approach has the drawback that a host needs to recognize all the interior gateway protocols that gateways may use (see "Requirements of Internet Gateways [RFC-1009]). In addition,

it only works on a broadcast network.

At present, pinging (i.e., using ICMP Echo messages) is the mechanism for gateway probing when absolutely required. A successful ping guarantees that the addressed interface and its associated machine are up, but it does not guarantee that the machine is a gateway as opposed to a host. The normal inference is that if a Redirect or other evidence indicates that a machine was a gateway, successful pings will indicate that the machine is still up and hence still a gateway. However, since a host silently discards packets that a gateway would forward or redirect, this assumption could sometimes fail. To avoid this problem, a new ICMP message under development will ask "are you a gateway?"

Implementing Dead Gateway Detection

The following specific algorithm has been suggested:

- Associate a "reroute timer" with each gateway pointed to by the route cache. Initialize the timer to a value Tr, which must be small enough to allow detection of a dead gateway before transport connections time out.
- Positive advice would reset the reroute timer to Tr. Negative advice would reduce or zero the reroute timer.
- Whenever the IP layer used a particular gateway to route a datagram, it would check the corresponding reroute timer. If the timer had expired (reached zero), the IP layer would send a ping to the gateway, followed immediately by the datagram.
- The ping (ICMP Echo) would be sent again if necessary, up to N times. If no ping reply was received in N tries, the gateway would be assumed to have failed, and a new first-hop gateway would be chosen for all cache entries pointing to the failed gateway.

Note that the size of Tr is inversely related to the amount of advice available. Tr should be large enough to insure that:

* Any pinging will be at a low level (e.g., <10%) of all packets sent to a gateway from the host, AND
* pinging is infrequent (e.g., every 3 minutes)

Since the recommended algorithm is concerned with the gateways pointed to by route cache entries, rather than the cache entries themselves, a two level data structure (perhaps coordinated with ARP or similar caches) may be desirable for implementing a route cache.

New Gateway Selection

If the failed gateway is not the current default, the IP layer can immediately switch to a default gateway. If it is the current default that failed, the IP layer **must** select a different default gateway (assuming more than one default is known) for the failed route and for establishing new routes.

Discussion

When a gateway does fail, the other gateways on the connected network will learn of the failure through some inter-gateway routing protocol. However, this will not happen instantaneously, since gateway routing protocols typically have a settling time of 30-60 seconds. If the host switches to an alternative gateway before the gateways have agreed on the failure, the new target gateway will probably forward the datagram to the failed gateway and send a Redirect back to the host pointing to the failed gateway (!). The result is likely to be a rapid oscillation in the contents of the host's route cache during the gateway settling period. It has been proposed that the dead- gateway logic should include some hysteresis mechanism to prevent such oscillations. However, experience has not shown any harm from such oscillations, since service cannot be restored to the host until the gateways' routing information does settle down.

Implementation

One implementation technique for choosing a new default gateway is to simply round-robin among the default gateways in the host's list. Another is to rank the gateways in priority order, and when the current default gateway is not the highest priority one, to "ping" the higher-priority gateways slowly to detect when they return to service. This pinging can be at a very low rate, e.g., 0.005 per second.

Source Route Forwarding

Subject to restrictions given below, a host **may** be able to act as an intermediate hop in a source route, forwarding a source- routed datagram to the next specified hop.

However, in performing this gateway-like function, the host **must** obey all the relevant rules for a gateway forwarding source-routed datagrams (see "Requirements for Internet Gateways [RFC-1009]). This includes the following specific provisions, which override the corresponding host provisions given earlier in this document:

(A) TTL
The TTL field **must** be decremented and the datagram perhaps discarded as specified for a gateway in "Requirements for Internet Gateways [RFC-1009].

(B) ICMP Destination Unreachable
A host **must** be able to generate Destination Unreachable messages with the following codes:

4 (Fragmentation Required but DF Set) when a source- routed datagram cannot be fragmented to fit into the target network;

5 (Source Route Failed) when a source-routed datagram cannot be forwarded, e.g., because of a routing problem or because the next hop of a strict source route is not on a connected network.

(C) IP Source Address
A source-routed datagram being forwarded **may** (and normally will) have a source address that is not one of the IP addresses of the forwarding host.

(D) Record Route Option
A host that is forwarding a source-routed datagram containing a Record Route option **must** update that option, if it has room.

(E) Timestamp Option
A host that is forwarding a source-routed datagram containing a Timestamp Option **must** add the current timestamp to that option, according to the rules for this option.

To define the rules restricting host forwarding of source- routed datagrams, we use the term "local source-routing" if the next hop will be through the same physical interface through which the datagram arrived; otherwise, it is "non-local source-routing".

- A host is permitted to perform local source-routing without restriction.
- A host that supports non-local source-routing **must** have a configurable switch to disable forwarding, and this switch **must** default to disabled.
- The host **must** satisfy all gateway requirements for configurable policy filters restricting non- local forwarding as described in "Requirements for Internet Gateways [RFC-1009].

If a host receives a datagram with an incomplete source route but does not forward it for some reason, the host **should** return an ICMP Destination Unreachable (code 5, Source Route Failed) message, unless the datagram was itself an ICMP error message.

Initialization

The following information **must** be configurable:

(1) IP address(es).

(2) Address mask(s).

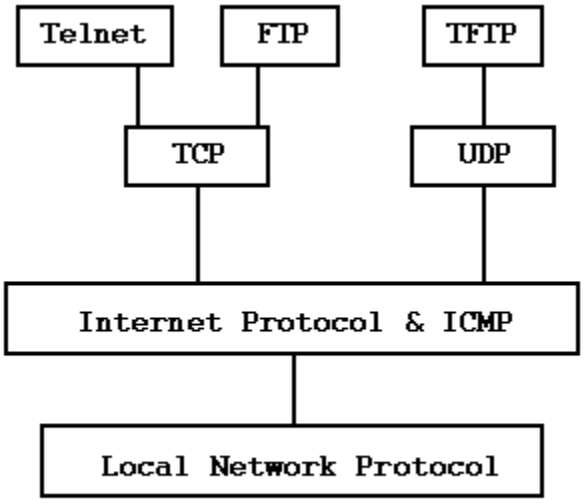
(3) A list of default gateways, with a preference level.

A manual method of entering this configuration data **must** be provided. In addition, a variety of methods can be used to determine this information dynamically.

Discussion
Some host implementations use "wiretapping" of gateway protocols on a broadcast network to learn what gateways exist. A standard method for default gateway discovery is under development.

Relation to Other Protocols

The following diagram illustrates the place of the internet protocol in the protocol hierarchy:



Protocol Relationships

Internet protocol interfaces on one side to the higher level host-to-host protocols and on the other side to the local network protocol. In this context a "local network" may be a small network in a building or a large network such as the ARPANET.

RFC-791 Internet Protocol - Overview

Model of Operation

The model of operation for transmitting a datagram from one application program to another is illustrated by the following scenario:

We suppose that this transmission will involve one intermediate gateway.

The sending application program prepares its data and calls on its local internet module to send that data as a datagram and passes the destination address and other parameters as arguments of the call.

The internet module prepares a datagram header and attaches the data to it. The internet module determines a local network address for this internet address, in this case it is the address of a gateway.

It sends this datagram and the local network address to the local network interface.

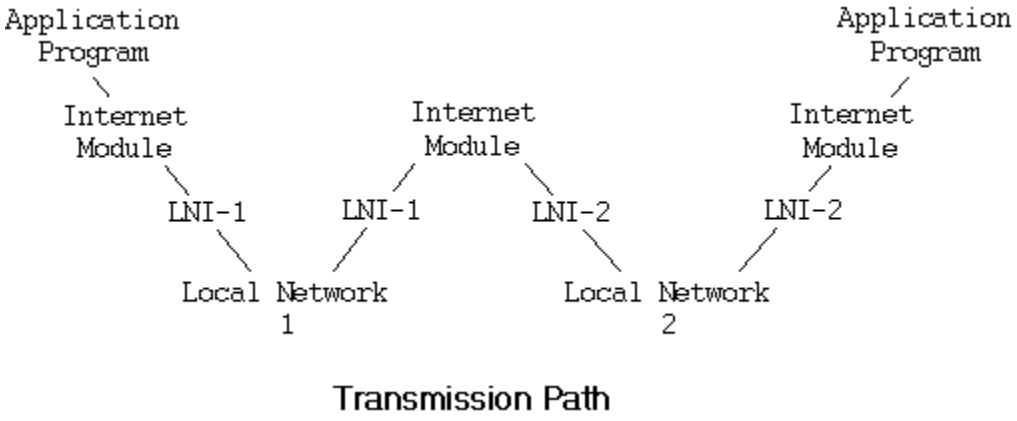
The local network interface creates a local network header, and attaches the datagram to it, then sends the result via the local network.

The datagram arrives at a gateway host wrapped in the local network header, the local network interface strips off this header, and turns the datagram over to the internet module. The internet module determines from the internet address that the datagram is to be forwarded to another host in a second network. The internet module determines a local net address for the destination host. It calls on the local network interface for that network to send the datagram.

This local network interface creates a local network header and attaches the datagram sending the result to the destination host.

At this destination host the datagram is stripped of the local net header by the local network interface and handed to the internet module.

The internet module determines that the datagram is for an application program in this host. It passes the data to the application program in response to a system call, passing the source address and other parameters as results of the call.



RFC-791 Internet Protocol - Overview

Function Description

The function or purpose of Internet Protocol is to move datagrams through an interconnected set of networks. This is done by passing the datagrams from one internet module to another until the destination is reached. The internet modules reside in hosts and gateways in the internet system. The datagrams are routed from one internet module to another through individual networks based on the interpretation of an internet address. Thus, one important mechanism of the internet protocol is the internet address.

In the routing of messages from one internet module to another, datagrams may need to traverse a network whose maximum packet size is smaller than the size of the datagram. To overcome this difficulty, a fragmentation mechanism is provided in the internet protocol.

Addressing

A distinction is made between names, addresses, and routes as discussed in "Names, Addresses, Ports, and Routes" [RFC-814]. A name indicates what we seek. An address indicates where it is. A route indicates how to get there. The internet protocol deals primarily with addresses. It is the task of higher level (i.e., host-to-host or application) protocols to make the mapping from names to addresses. The internet module maps internet addresses to local net addresses. It is the task of lower level (i.e., local net or gateways) procedures to make the mapping from local net addresses to routes.

Addresses are fixed length of four octets (32 bits). An address begins with a network number, followed by local address (called the "rest" field). There are three formats or classes of internet addresses: in class a, the high order bit is zero, the next 7 bits are the network, and the last 24 bits are the local address; in class b, the high order two bits are one-zero, the next 14 bits are the network and the last 16 bits are the local address; in class c, the high order three bits are one-one-zero, the next 21 bits are the network and the last 8 bits are the local address.

Care must be taken in mapping internet addresses to local net addresses; a single physical host must be able to act as if it were several distinct hosts to the extent of using several distinct internet addresses. Some hosts will also have several physical interfaces (multi-homing).

That is, provision must be made for a host to have several physical interfaces to the network with each having several logical internet addresses.

Examples of address mappings may be found in "Address Mappings" RFC-796.

Local Multihoming Introduction

A multihomed host has multiple IP addresses, which we may think of as "logical interfaces". These logical interfaces may be associated with one or more physical interfaces, and these physical interfaces may be connected to the same or different networks.

Here are some important cases of multihoming:

(a) Multiple Logical Networks

The Internet architects envisioned that each physical network would have a single unique IP network (or subnet) number. However, LAN administrators have sometimes found it useful to violate this assumption, operating a LAN with multiple logical networks per physical connected network.

If a host connected to such a physical network is configured to handle traffic for each of N different logical networks, then the host will have N logical interfaces. These could share a single physical interface, or might use N physical interfaces to the same network.

(b) Multiple Logical Hosts

When a host has multiple IP addresses that all have the same <Network-number> part (and the same <Subnet- number> part, if any), the logical interfaces are known as "logical hosts". These logical interfaces might share a single physical interface or might use separate physical interfaces to the same physical network.

(c) Simple Multihoming

In this case, each logical interface is mapped into a separate physical interface and each physical interface is connected to a different physical network. The term "multihoming" was originally applied only to this case, but it is now applied more generally.

A host with embedded gateway functionality will typically fall into the simple multihoming case. Note, however, that a host may be simply multihomed without containing an embedded gateway, i.e., without forwarding datagrams from one connected network to another.

This case presents the most difficult routing problems. The choice of interface (i.e., the choice of first-hop network) may significantly affect performance or even reachability of remote parts of the Internet.

Finally, we note another possibility that is NOT multihoming: one logical interface may be bound to multiple physical interfaces, in order to increase the reliability or throughput between directly connected machines by providing alternative physical paths between them. For instance, two systems might be connected by multiple point-to-point links. We call this "link-layer multiplexing". With link-layer multiplexing, the protocols above the link layer are unaware that multiple physical interfaces are present; the link- layer device driver is responsible for multiplexing and routing packets across the physical interfaces.

In the Internet protocol architecture, a transport protocol instance ("entity") has no address of its own, but instead uses a single Internet Protocol (IP) address. This has implications for the IP, transport, and application layers, and for the interfaces between them. In particular,

the application software may have to be aware of the multiple IP addresses of a multihomed host; in other cases, the choice can be made within the network software.

Requirements

The following general rules apply to the selection of an IP source address for sending a datagram from a multihomed host.

(1) if the datagram is sent in response to a received datagram, the source address for the response **should** be the specific-destination address of the request. See UDP Multihoming and TCP Multihoming and **Applications on Multihomed Hosts** for more specific requirements on higher layers.

Otherwise, a source address must be selected.

(2) an application **must** be able to explicitly specify the source address for initiating a connection or a request.

(3) an the absence of such a specification, the networking software **must** choose a source address. Rules for this choice are described below.

There are two key requirement issues related to multihoming:

(A) a host **may** silently discard an incoming datagram whose destination address does not correspond to the physical interface through which it is received.

(B) a host **may** restrict itself to sending (non-source- routed) IP datagrams only through the physical interface that corresponds to the IP source address of the datagrams.

Discussion

Internet host implementors have used two different conceptual models for multihoming, briefly summarized in the following discussion. This document takes no stand on which model is preferred; each seems to have a place. This ambivalence is reflected in the issues (A) and (B) being optional.

Strong ES Model

The Strong ES (End System, i.e., host) model emphasizes the host/gateway (ES/IS) distinction, and would therefore substitute **must** for **may** in issues (A) and (B) above. It tends to model a multihomed host as a set of logical hosts within the same physical host.

With respect to (A), proponents of the Strong ES model note that automatic Internet routing mechanisms could not route a datagram to a physical interface that did not correspond to the destination address.

Under the Strong ES model, the route computation for an outgoing datagram is the mapping:

route(src IP addr, dest IP addr, TOS) -> gateway

Here the source address is included as a parameter in order to select a gateway that is directly reachable on the corresponding physical interface. Note that this model logically requires that in general there be at least one default gateway, and preferably multiple defaults, for each IP source address.

Weak ES Model

This view de-emphasizes the ES/IS distinction, and would therefore substitute **must** NOT for **may** in issues (A) and (B). This model may be the more natural one for hosts that wiretap gateway routing protocols, and is necessary for hosts that have embedded gateway functionality.

The Weak ES Model may cause the Redirect mechanism to fail. If a datagram is sent out a physical interface that does not correspond to the destination address, the first-hop gateway will not realize when it needs to send a Redirect. On the other hand, if the host has embedded gateway functionality, then it has routing information without listening to Redirects.

In the Weak ES model, the route computation for an outgoing datagram is the mapping:

route(dest IP addr, TOS) -> gateway, interface

Choosing a Source Address

Discussion

When it sends an initial connection request (e.g., a TCP "SYN" segment) or a datagram service request (e.g., a UDP-based query), the transport layer on a multihomed host needs to know which source address to use. If the application does not specify it, the transport layer must ask the IP layer to perform the conceptual mapping:

GET_SRCADDR(remote IP addr, TOS) -> local IP address

Here TOS is the Type-of-Service value (see Type-of-Service), and the result is the desired source address. The following rules are suggested for implementing this mapping:

(a) If the remote Internet address lies on one of the (sub-) nets to which the host is directly connected, a corresponding source address may be chosen, unless the corresponding interface is known to be down.

(b) The route cache may be consulted, to see if there is an active route to the specified destination network through any network interface; if so, a local IP address corresponding to that interface may be chosen.

(c) The table of static routes, if any (see Gateway Selection) may be similarly consulted.

(d) The default gateways may be consulted. If these gateways are assigned to different interfaces, the interface corresponding to the gateway with the highest preference may be chosen.

In the future, there may be a defined way for a multihomed host to ask the gateways on all connected networks for advice about the best network to use for a given destination.

Implementation

It will be noted that this process is essentially the same as datagram routing (see Routing Outbound Datagrams), and therefore hosts may be able to combine the implementation of the two functions.

Fragmentation

Fragmentation of an internet datagram is necessary when it originates in a local net that allows a large packet size and must traverse a local net that limits packets to a smaller size to reach its destination.

An internet datagram can be marked "don't fragment." Any internet datagram so marked is not to be internet fragmented under any circumstances. If internet datagram marked don't fragment cannot be delivered to its destination without fragmenting it, it is to be discarded instead.

Fragmentation, transmission and reassembly across a local network which is invisible to the internet protocol module is called intranet fragmentation and may be used [6].

The internet fragmentation and reassembly procedure needs to be able to break a datagram into an almost arbitrary number of pieces that can be later reassembled. The receiver of the fragments uses the identification field to ensure that fragments of different datagrams are not mixed. The fragment offset field tells the receiver the position of a fragment in the original datagram. The fragment offset and length determine the portion of the original datagram covered by this fragment. The more-fragments flag indicates (by being reset) the last fragment. These fields provide sufficient information to reassemble datagrams.

The identification field is used to distinguish the fragments of one datagram from those of another. The originating protocol module of an internet datagram sets the identification field to a value that must be unique for that source-destination pair and protocol for the time the datagram will be active in the internet system. The originating protocol module of a complete datagram sets the more-fragments flag to zero and the fragment offset to zero.

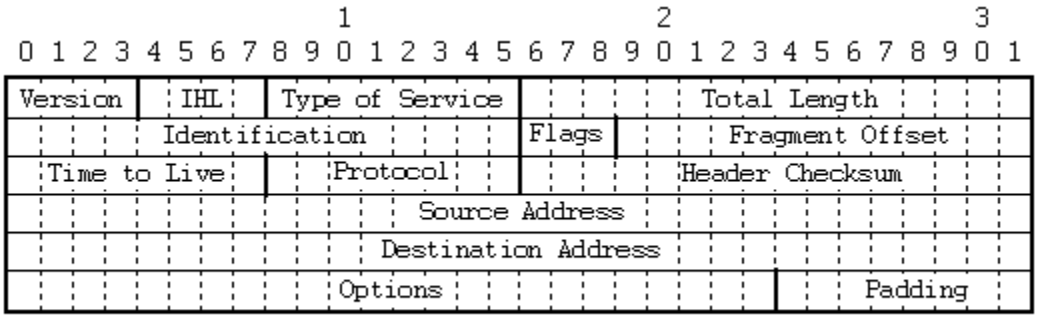
To fragment a long internet datagram, an internet protocol module (for example, in a gateway), creates two new internet datagrams and copies the contents of the internet header fields from the long datagram into both new internet headers. The data of the long datagram is divided into two portions on a 8 octet (64 bit) boundary (the second portion might not be an integral multiple of 8 octets, but the first must be). Call the number of 8 octet blocks in the first portion NFB (for Number of Fragment Blocks). The first portion of the data is placed in the first new internet datagram, and the total length field is set to the length of the first datagram. The more-fragments flag is set to one. The second portion of the data is placed in the second new internet datagram, and the total length field is set to the length of the second datagram. The more-fragments flag carries the same value as the long datagram. The fragment offset field of the second new internet datagram is set to the value of that field in the long datagram plus NFB.

This procedure can be generalized for an n-way split, rather than the two-way split described.

To assemble the fragments of an internet datagram, an internet protocol module (for example at a destination host) combines internet datagrams that all have the same value for the four fields: identification, source, destination, and protocol. The combination is done by placing the data portion of each fragment in the relative position indicated by the fragment offset in that fragment's internet header. The first fragment will have the fragment offset zero, and the last fragment will have the more-fragments flag reset to zero.

RFC-791 Internet Protocol - Specification

Internet Header Format



Example Internet Datagram Header

Use the Browse button to see a description of each header field.

RFC-791 Internet Protocol - Specification

Version: 4 bits

The Version field indicates the format of the internet header. This document describes version 4.

A datagram whose version number **is not** 4 **must** be silently discarded.

RFC-791 Internet Protocol - Specification

IHL: 4 bits

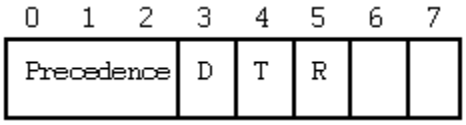
Internet Header Length is the length of the internet header in 32 bit words, and thus points to the beginning of the data. Note that the minimum value for a correct header is 5.

RFC-791 Internet Protocol - Specification

Type of Service: 8 bits

The Type of Service provides an indication of the abstract parameters of the quality of service desired. These parameters are to be used to guide the selection of the actual service parameters when transmitting a datagram through a particular network. Several networks offer service precedence, which somehow treats high precedence traffic as more important than other traffic (generally by accepting only traffic above a certain precedence at time of high load). The major choice is a three way tradeoff between low-delay, high-reliability, and high-throughput.

Bits 0-2: Precedence.
Bit 3: 0 = Normal Delay, 1 = Low Delay.
Bits 4: 0 = Normal Throughput, 1 = High Throughput.
Bits 5: 0 = Normal Relibility, 1 = High Relibility.
Bit 6-7: Reserved for Future Use.



Precedence

111 - Network Control
110 - Internetwork Control
101 - CRITIC/ECP
100 - Flash Override
011 - Flash
010 - Immediate
001 - Priority
000 - Routine

The use of the Delay, Throughput, and Reliability indications may increase the cost (in some sense) of the service. In many networks better performance for one of these parameters is coupled with worse performance on another. Except for very unusual cases at most two of these three indications should be set.

The type of service is used to specify the treatment of the datagram during its transmission through the internet system. Example mappings of the internet type of service to the actual service provided on networks such as AUTODIN II, ARPANET, SATNET, and PRNET are given in "Service Mappings" [RFC-795] though this document is **obsolete**.

The Network Control precedence designation is intended to be used within a network only. The actual use and control of that designation is up to each network. The Internetwork Control designation is intended for use by gateway control originators only. If the actual use of these precedence designations is of concern to a particular network, it is the responsibility of that network to control the access to, and use of, those precedence designations.

RFC-791 Internet Protocol - Specification

Total Length: 16 bits

Total Length is the length of the datagram, measured in octets, including internet header and data. This field allows the length of a datagram to be up to 65,535 octets. Such long datagrams are impractical for most hosts and networks. All hosts must be prepared to accept datagrams of up to 576 octets (whether they arrive whole or in fragments). It is recommended that hosts only send datagrams larger than 576 octets if they have assurance that the destination is prepared to accept the larger datagrams.

The number 576 is selected to allow a reasonable sized data block to be transmitted in addition to the required header information. For example, this size allows a data block of 512 octets plus 64 header octets to fit in a datagram. The maximal internet header is 60 octets, and a typical internet header is 20 octets, allowing a margin for headers of higher level protocols.

RFC-791 Internet Protocol - Specification

Identification: 16 bits

An identifying value assigned by the sender to aid in assembling the fragments of a datagram.

Flags: 3 bits

Various Control Flags.

Bit 0: reserved, must be zero
Bit 1: (DF) 0 = May Fragment, 1 = Don't Fragment.
Bit 2: (MF) 0 = Last Fragment, 1 = More Fragments.

Fragment Offset: 13 bits

This field indicates where in the datagram this fragment belongs.

The fragment offset is measured in units of 8 octets (64 bits). The first fragment has offset zero.

Time to Live: 8 bits

This field indicates the maximum time the datagram is allowed to remain in the internet system. If this field contains the value zero, then the datagram must be destroyed. This field is modified in internet header processing. The time is measured in units of seconds, but since every module that processes a datagram must decrease the TTL by at least one even if it process the datagram in less than a second, the TTL must be thought of only as an upper bound on the time a datagram may exist. The intention is to cause undeliverable datagrams to be discarded, and to bound the maximum datagram lifetime.

Protocol: 8 bits

This field indicates the next level protocol used in the data portion of the internet datagram. The values for various protocols are specified in "Assigned Numbers".

Header Checksum: 16 bits

A checksum on the header only. Since some header fields change (e.g., time to live), this is recomputed and verified at each point that the internet header is processed.

The checksum algorithm is:

The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header. For purposes of computing the checksum, the value of the checksum field is zero.

This is a simple to compute checksum and experimental evidence indicates it is adequate, but it is provisional and may be replaced by a CRC procedure, depending on further experience.

Source Address: 32 bits

The source address. See Discussion.

Destination Address: 32 bits

The destination address. See Discussion.

Options: variable

The options may appear or not in datagrams. They must be implemented by all IP modules (host and gateways). What is optional is their transmission in any particular datagram, not their implementation.

In some environments the security option may be required in all datagrams.

The option field is variable in length. There may be zero or more options. There are two cases for the format of an option:

Case 1: A single octet of option-type.

Case 2: An option-type octet, an option-length octet, and the actual option-data octets.

The option-length octet counts the option-type octet and the option-length octet as well as the option-data octets.

The option-type octet is viewed as having 3 fields:
1 bit copied flag,
2 bits option class,
5 bits option number.

The copied flag indicates that this option is copied into all fragments on fragmentation.

0 = not copied
1 = copied

The option classes are:

0 = control
1 = reserved for future use
2 = debugging and measurement
3 = reserved for future use

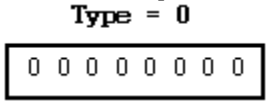
The following internet options are defined:

CLASS	NUMBER	LENGTH	DESCRIPTION
0	0	-	End of Option list. This option occupies only 1 octet; it has no length octet.
0	1	-	No Operation. This option occupies only 1 octet; it has no length octet.
0	2	11	Security. Used to carry Security, Compartmentation, User Group (TCC), and Handling Restriction Codes compatible with DOD requirements.
0	3	var.	Loose Source Routing. Used to route the internet datagram based on information supplied by the source.
0	9	var.	Strict Source Routing. Used to route the internet datagram based on information supplied by the source.
0	7	var.	Record Route. Used to trace the route an internet datagram takes.
0	8	4	Stream ID. Used to carry the stream

			identifier.
2	4	var.	Internet Timestamp.

RFC-791 Internet Protocol - Specific Option Definitions

End of Option List

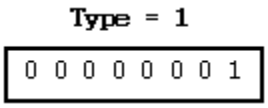


This option indicates the end of the option list. This might not coincide with the end of the internet header according to the internet header length. This is used at the end of all options, not the end of each option, and need only be used if the end of the options would not otherwise coincide with the end of the internet header.

May be copied, introduced, or deleted on fragmentation, or for any other reason.

RFC-791 Internet Protocol - Specific Option Definitions

No Operation



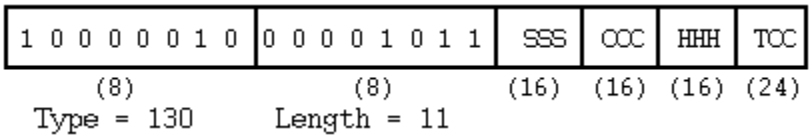
This option may be used between options, for example, to align the beginning of a subsequent option on a 32 bit boundary.

May be copied, introduced, or deleted on fragmentation, or for any other reason.

RFC-791 Internet Protocol - Specific Option Definitions

Security

This option provides a way for hosts to send security, compartmentation, handling restrictions, and TCC (closed user group) parameters. The format for this option as originally proposed is as follows (**Note: These options are obsolete)**:



Security (S field): 16 bits

Specifies one of 16 levels of security (eight of which are reserved for future use).

00000000 00000000 - Unclassified
11110001 00110101 - Confidential
01111000 10011010 - EFTO
10111100 01001101 - MMMM
01011110 00100110 - PROG
10101111 00010011 - Restricted
11010111 10001000 - Secret
01101011 11000101 - Top Secret
00110101 11100010 - (Reserved for future use)
10011010 11110001 - (Reserved for future use)
01001101 01111000 - (Reserved for future use)
00100100 10111101 - (Reserved for future use)
00010011 01011110 - (Reserved for future use)
10001001 10101111 - (Reserved for future use)
11000100 11010110 - (Reserved for future use)
11100010 01101011 - (Reserved for future use)

Compartments (C field): 16 bits

> An all zero value is used when the information transmitted is not compartmented. Other values for the compartments field may be obtained from the Defense Intelligence Agency.

Handling Restrictions (H field): 16 bits

> The values for the control and release markings are alphanumeric digraphs and are defined in the Defense Intelligence Agency Manual DIAM 65-19, "Standard Security Markings".

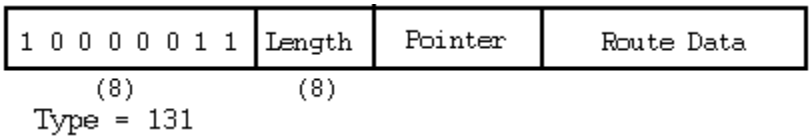
Transmission Control Code (TCC field): 24 bits

> Provides a means to segregate traffic and define controlled communities of interest among subscribers. The TCC values are trigraphs, and are available from HQ DCA Code 530.

Must be copied on fragmentation. This option appears at most once in a datagram.

RFC-791 Internet Protocol - Specific Option Definitions

Loose Source and Record Route



The loose source and record route (LSRR) option provides a means for the source of an internet datagram to supply routing information to be used by the gateways in forwarding the datagram to the destination, and to record the route information.

The option begins with the option type code. The second octet is the option length which includes the option type code and the length octet, the pointer octet, and length-3 octets of route data. The third octet is the pointer into the route data indicating the octet which begins the next source address to be processed. The pointer is relative to this option, and the smallest legal value for the pointer is 4.

A route data is composed of a series of internet addresses. Each internet address is 32 bits or 4 octets. If the pointer is greater than the length, the source route is empty (and the recorded route full) and the routing is to be based on the destination address field.

If the address in destination address field has been reached and the pointer is not greater than the length, the next address in the source route replaces the address in the destination address field, and the recorded route address replaces the source address just used, and pointer is increased by four.

The recorded route address is the internet module's own internet address as known in the environment into which this datagram is being forwarded.

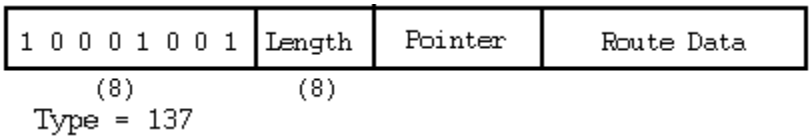
This procedure of replacing the source route with the recorded route (though it is in the reverse of the order it must be in to be used as a source route) means the option (and the IP header as a whole) remains a constant length as the datagram progresses through the internet.

This option is a loose source route because the gateway or host IP is allowed to use any route of any number of other intermediate gateways to reach the next address in the route.

Must be copied on fragmentation. Appears at most once in a datagram.

RFC-791 Internet Protocol - Specific Option Definitions

Strict Source and Record Route



The strict source and record route (SSRR) option provides a means for the source of an internet datagram to supply routing information to be used by the gateways in forwarding the datagram to the destination, and to record the route information.

The option begins with the option type code. The second octet is the option length which includes the option type code and the length octet, the pointer octet, and length-3 octets of route data. The third octet is the pointer into the route data indicating the octet which begins the next source address to be processed. The pointer is relative to this option, and the smallest legal value for the pointer is 4.

A route data is composed of a series of internet addresses. Each internet address is 32 bits or 4 octets. If the pointer is greater than the length, the source route is empty (and the recorded route full) and the routing is to be based on the destination address field.

If the address in destination address field has been reached and the pointer is not greater than the length, the next address in the source route replaces the address in the destination address field, and the recorded route address replaces the source address just used, and pointer is increased by four.

The recorded route address is the internet module's own internet address as known in the environment into which this datagram is being forwarded.

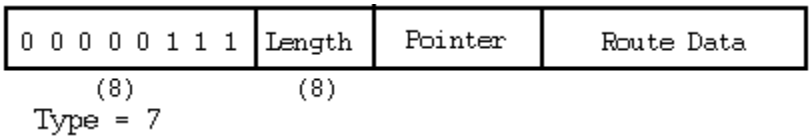
This procedure of replacing the source route with the recorded route (though it is in the reverse of the order it must be in to be used as a source route) means the option (and the IP header as a whole) remains a constant length as the datagram progresses through the internet.

This option is a strict source route because the gateway or host IP must send the datagram directly to the next address in the source route through only the directly connected network indicated in the next address to reach the next gateway or host specified in the route.

Must be copied on fragmentation. Appears at most once in a datagram.

RFC-791 Internet Protocol - Specific Option Definitions

Record Route



The record route option provides a means to record the route of an internet datagram.

The option begins with the option type code. The second octet is the option length which includes the option type code and the length octet, the pointer octet, and length-3 octets of route data. The third octet is the pointer into the route data indicating the octet which begins the next area to store a route address. The pointer is relative to this option, and the smallest legal value for the pointer is 4.

A recorded route is composed of a series of internet addresses. Each internet address is 32 bits or 4 octets. If the pointer is greater than the length, the recorded route data area is full. The originating host must compose this option with a large enough route data area to hold all the address expected. The size of the option does not change due to adding addresses. The intitial contents of the route data area must be zero.

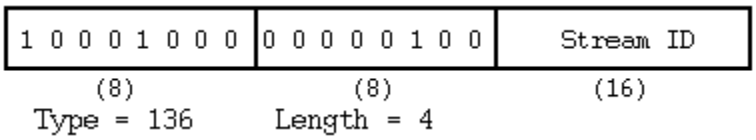
When an internet module routes a datagram it checks to see if the record route option is present. If it is, it inserts its own internet address as known in the environment into which this datagram is being forwarded into the recorded route begining at the octet indicated by the pointer, and increments the pointer by four.

If the route data area is already full (the pointer exceeds the length) the datagram is forwarded without inserting the address into the recorded route. If there is some room but not enough room for a full address to be inserted, the original datagram is considered to be in error and is discarded. In either case an ICMP parameter problem message may be sent to the source host.

Not copied on fragmentation, goes in first fragment only. Appears at most once in a datagram.

RFC-791 Internet Protocol - Specific Option Definitions

Stream Identifier



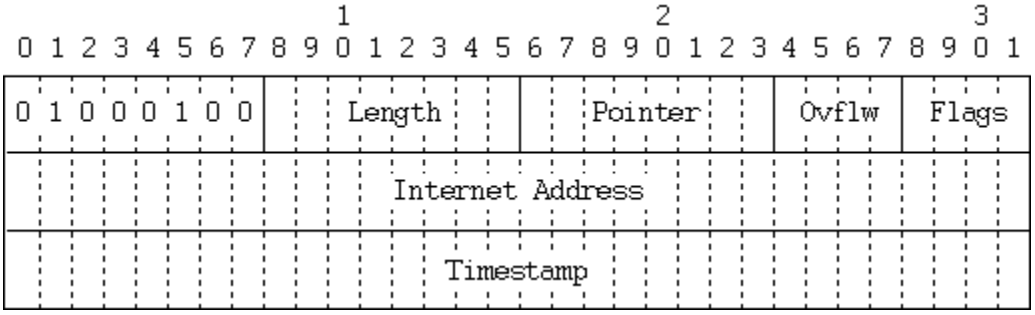
This option provides a way for the 16-bit SATNET stream identifier to be carried through networks that do not support the stream concept.

Must be copied on fragmentation. Appears at most once in a datagram.

This option is obsolete and should not be sent.

RFC-791 Internet Protocol - Specific Option Definitions

Internet Timestamp



The Option Length is the number of octets in the option counting the type, length, pointer, and overflow/flag octets (maximum length 40).

The Pointer is the number of octets from the beginning of this option to the end of timestamps plus one (i.e., it points to the octet beginning the space for next timestamp). The smallest legal value is 5. The timestamp area is full when the pointer is greater than the length.

The Overflow (oflw) [4 bits] is the number of IP modules that cannot register timestamps due to lack of space.

The Flag (flg) [4 bits] values are

0	time stamps only, stored in consecutive 32-bit words,
1	each timestamp is preceded with internet address of the registering entity,
3	the internet address fields are prespecified. An IP module only registers its timestamp if it matches its own address with the next specified internet address.

The Timestamp is a right-justified, 32-bit timestamp in milliseconds since midnight UT. If the time is not available in milliseconds or cannot be provided with respect to midnight UT then any time may be inserted as a timestamp provided the high order bit of the timestamp field is set to one to indicate the use of a non-standard value.

The originating host must compose this option with a large enough timestamp data area to hold all the timestamp information expected. The size of the option does not change due to adding timestamps. The intitial contents of the timestamp data area must be zero or internet address/zero pairs.

If the timestamp data area is already full (the pointer exceeds the length) the datagram is forwarded without inserting the timestamp, but the overflow count is incremented by one.

If there is some room but not enough room for a full timestamp to be inserted, or the overflow count itself overflows, the original datagram is considered to be in error and is discarded. In either case an ICMP parameter problem message may be sent to the source host.

The timestamp option is not copied upon fragmentation. It is carried in the first fragment. Appears at most once in a datagram.

Padding: variable

The internet header padding is used to ensure that the internet header ends on a 32 bit boundary. The padding is zero.

Discussion

The implementation of a protocol must be robust. Each implementation must expect to interoperate with others created by different individuals. While the goal of this specification is to be explicit about the protocol there is the possibility of differing interpretations. In general, an implementation must be conservative in its sending behavior, and liberal in its receiving behavior. That is, it must be careful to send well-formed datagrams, but must accept any datagram that it can interpret (e.g., not object to technical errors where the meaning is still clear).

The basic internet service is datagram oriented and provides for the fragmentation of datagrams at gateways, with reassembly taking place at the destination internet protocol module in the destination host. Of course, fragmentation and reassembly of datagrams within a network or by private agreement between the gateways of a network is also allowed since this is transparent to the internet protocols and the higher-level protocols. This transparent type of fragmentation and reassembly is termed "network-dependent" (or intranet) fragmentation and is not discussed further here.

Internet addresses distinguish sources and destinations to the host level and provide a protocol field as well. It is assumed that each protocol will provide for whatever multiplexing is necessary within a host.

RFC-791 Internet Protocol - Discussion

Addressing

To provide for flexibility in assigning address to networks and allow for the large number of small to intermediate sized networks the interpretation of the address field is coded to specify a small number of networks with a large number of host, a moderate number of networks with a moderate number of hosts, and a large number of networks with a small number of hosts. In addition there is an escape code for extended addressing mode. **Additions and Extensions**.

Address Formats:

High Order Bits	Format	Class
0	7 bits of net, 24 bits of host	A
10	14 bits of net, 16 bits of host	B
110	21 bits of net, 8 bits of host	C
111	escape to extended addressing mode	

A value of zero in the network field means this network. This is only used in certain ICMP messages. The extended addressing mode is undefined. Both of these features are reserved for future use.

The actual values assigned for network addresses is given in "Assigned Numbers".

The local address, assigned by the local network, must allow for a single physical host to act as several distinct internet hosts. That is, there must be a mapping between internet host addresses and network/host interfaces that allows several internet addresses to correspond to one interface. It must also be allowed for a host to have several physical interfaces and to treat the datagrams from several of them as if they were all addressed to a single host.

Address mappings between internet addresses and addresses for ARPANET, SATNET, PRNET, and other networks are described in "Address Mappings" [RFC-796].

See Additional Addressing Notes.

RFC-791 Internet Protocol - Addressing

Additional Notes

There are now five classes of IP addresses: Class A through Class E.

Class D addresses are used for IP multicasting [RFC-1112] while Class E addresses are reserved for experimental use.

A multicast (Class D) address is a 28-bit logical address that stands for a group of hosts, and may be either permanent or transient. Permanent multicast addresses are allocated by the Internet Assigned Number Authority and are listed in Assigned Numbers, while transient addresses may be allocated dynamically to transient groups. Group membership is determined dynamically using IGMP.

We now summarize the important special cases for Class A, B, and C IP addresses, using the following notation for an IP address:

{ <Network-number>, <Host-number> }
or
{ <Network-number>, <Subnet-number>, <Host-number> }

and the notation "-1" for a field that contains all 1 bits. This notation is not intended to imply that the 1-bits in an address mask need be contiguous.

(a) { 0, 0 }

This host on this network. **must not** be sent, except as a source address as part of an initialization procedure by which the host learns its own IP address.

See also Broadcasts for a non-standard use of {0,0}.

(b) { 0, <Host-number> }

Specified host on this network. It **must not** be sent, except as a source address as part of an initialization procedure by which the host learns its full IP address.

(c) { -1, -1 }

Limited broadcast. It **must not** be used as a source address.

A datagram with this destination address will be received by every host on the connected physical network but will not be forwarded outside that network.

(d) { <Network-number>, -1 }

Directed broadcast to the specified network. It **must not** be used as a source address.

(e) { <Network-number>, <Subnet-number>, -1 }

Directed broadcast to the specified subnet. It **must not** be used as a source address.

(f) { <Network-number>, -1, -1 }

Directed broadcast to all subnets of the specified subnetted network. It **must not** be used as a source address.

(g) { 127, <any> }

Internal host loopback address. Addresses of this form **must not** appear outside a host.

The <Network-number> is administratively assigned so that its value will be unique in the entire world.

IP addresses are not permitted to have the value 0 or -1 for any of the <Host-number>, <Network-number>, or <Subnet- number> fields (except in the special cases listed above). This implies that each of these fields will be at least two bits long.

For further discussion of broadcast addresses, see Broadcasts and Broadcasting Internet Datagrams in the Presence of Subnets [RFC-922].

A host **must** support the subnet extensions to IP [RFC-950]. As a result, there will be an address mask of the form: {-1, -1, 0} associated with each of the host's local IP addresses; see Address Mask and Local/Remote Decision.

When a host sends any datagram, the IP source address **must** be one of its own IP addresses (but not a broadcast or multicast address).

A host **must** silently discard an incoming datagram that is not destined for the host. An incoming datagram is destined for the host if the datagram's destination address field is:

(1) (one of) the host's IP address(es); or
(2) an IP broadcast address valid for the connected network; or
(3) the address for a multicast group of which the host is a member on the incoming physical interface.

For most purposes, a datagram addressed to a broadcast or multicast destination is processed as if it had been addressed to one of the host's IP addresses; we use the term "specific-destination address" for the equivalent local IP address of the host. The specific-destination address is defined to be the destination address in the IP header unless the header contains a broadcast or multicast address, in which case the specific-destination is an IP address assigned to the physical interface on which the datagram arrived.

A host **must** silently discard an incoming datagram containing an IP source address that is invalid by the rules of this section. This validation could be done in either the IP layer or by each protocol in the transport layer.

Discussion

A mis-addressed datagram might be caused by a link- layer broadcast of a unicast datagram or by a gateway or host that is confused or mis-configured.

An architectural goal for Internet hosts was to allow IP addresses to be featureless 32-bit numbers, avoiding algorithms that required a knowledge of the IP address format. Otherwise, any future change in the format or interpretation of IP addresses will require host software changes. However, validation of broadcast and multicast addresses violates this goal; a few other violations are described elsewhere in this document.

Implementers should be aware that applications depending upon the all-subnets directed broadcast address (f) may be unusable on some networks. All- subnets broadcast is not widely implemented in vendor gateways at present, and even when it is implemented, a particular network administration may disable it in the gateway configuration.

Broadcasts

There are four standard IP broadcast address forms (See Definition and Broadcasting Internet Datagrams in the Presence of Subnets [RFC-922]):

Limited Broadcast: {-1, -1}
Directed Broadcast: {<Network-number>,-1}
Subnet Directed Broadcast: {<Network-number>,<Subnet-number>,-1}
All-Subnets Directed Broadcast: {<Network-number>,-1,-1}

A host **must** recognize any of these forms in the destination address of an incoming datagram.

There is a class of hosts that use non-standard broadcast address forms, substituting 0 for -1. All hosts **should** recognize and accept any of these non-standard broadcast addresses as the destination address of an incoming datagram. A host **may** optionally have a configuration option to choose the 0 or the -1 form of broadcast address, for each physical interface, but this option **should** default to the standard (-1) form.

When a host sends a datagram to a link-layer broadcast address, the IP destination address **must** be a legal IP broadcast or IP multicast address.

A host **should** silently discard a datagram that is received via a link-layer broadcast but does not specify an IP multicast or broadcast destination address.

Hosts **should** use the Limited Broadcast address to broadcast to a connected network.

Discussion
Using the Limited Broadcast address instead of a Directed Broadcast address may improve system robustness. Problems are often caused by machines that do not understand the plethora of broadcast addresses, or that may have different ideas about which broadcast addresses are in use. The prime example of the latter is machines that do not understand subnetting but are attached to a subnetted net. Sending a Subnet Broadcast for the connected network will confuse those machines, which will see it as a message to some other host.

There has been discussion on whether a datagram addressed to the Limited Broadcast address ought to be sent from all the interfaces of a multihomed host. This specification takes no stand on the issue.

Fragmentation and Reassembly

The internet identification field (ID) is used together with the source and destination address, and the protocol fields, to identify datagram fragments for reassembly.

The More Fragments flag bit (MF) is set if the datagram is not the last fragment. The Fragment Offset field identifies the fragment location, relative to the beginning of the original unfragmented datagram. Fragments are counted in units of 8 octets. The fragmentation strategy is designed so than an unfragmented datagram has all zero fragmentation information (MF = 0, fragment offset = 0). If an internet datagram is fragmented, its data portion must be broken on 8 octet boundaries.

This format allows 2**13 = 8192 fragments of 8 octets each for a total of 65,536 octets. Note that this is consistent with the the datagram total length field (of course, the header is counted in the total length and not in the fragments).

When fragmentation occurs, some options are copied, but others remain with the first fragment only.

Every internet module must be able to forward a datagram of 68 octets without further fragmentation. This is because an internet header may be up to 60 octets, and the minimum fragment is 8 octets.

Every internet destination must be able to receive a datagram of 576 octets either in one piece or in fragments to be reassembled.

The fields which may be affected by fragmentation include:

(1) options field
(2) more fragments flag
(3) fragment offset
(4) internet header length field
(5) total length field
(6) header checksum

If the Don't Fragment flag (DF) bit is set, then internet fragmentation of this datagram is NOT permitted, although it may be discarded. This can be used to prohibit fragmentation in cases where the receiving host does not have sufficient resources to reassemble internet fragments.

One example of use of the Don't Fragment feature is to down line load a small host. A small host could have a boot strap program that accepts a datagram stores it in memory and then executes it.

The fragmentation and reassembly procedures are most easily described by examples. The following procedures are example implementations.

General notation in the following pseudo programs: "=<" means "less than or equal", "#" means "not equal", "=" means "equal", "<-" means "is set to". Also, "x to y" includes x and excludes y; for example, "4 to 7" would include 4, 5, and 6 (but not 7).

Fragmentation Procedure

RFC-791 Internet Protocol - Fragmentation Procedure

Overview

Optionally, the IP layer **may** implement a mechanism to fragment outgoing datagrams intentionally.

We designate by EMTU_S ("Effective MTU for sending") the maximum IP datagram size that may be sent, for a particular combination of IP source and destination addresses and perhaps TOS.

A host **must** implement a mechanism to allow the transport layer to learn MMS_S, the maximum transport-layer message size that may be sent for a given {source, destination, TOS} triplet (see GET_MAXSIZES call in Internet/Transport Layer Interface). If no local fragmentation is performed, the value of MMS_S will be:

MMS_S = EMTU_S - <IP header size>

and EMTU_S must be less than or equal to the MTU of the network interface corresponding to the source address of the datagram. Note that <IP header size> in this equation will be 20, unless the IP reserves space to insert IP options for its own purposes in addition to any options inserted by the transport layer.

A host that does not implement local fragmentation **must** ensure that the transport layer (for TCP) or the application layer (for UDP) obtains MMS_S from the IP layer and does not send a datagram exceeding MMS_S in size.

It is generally desirable to avoid local fragmentation and to choose EMTU_S low enough to avoid fragmentation in any gateway along the path. In the absence of actual knowledge of the minimum MTU along the path, the IP layer **should** use EMTU_S <= 576 whenever the destination address is not on a connected network, and otherwise use the connected network's MTU.

The MTU of each physical interface **must** be configurable.

A host IP layer implementation **may** have a configuration flag "All-Subnets-MTU", indicating that the MTU of the connected network is to be used for destinations on different subnets within the same network, but not for other networks. Thus, this flag causes the network class mask, rather than the subnet address mask, to be used to choose an EMTU_S. For a multihomed host, an "All-Subnets-MTU" flag is needed for each network interface.

Discussion

Picking the correct datagram size to use when sending data is a complex topic [IP:9].

(a) In general, no host is required to accept an IP datagram larger than 576 bytes (including header and data), so a host must not send a larger datagram without explicit knowledge or prior arrangement with the destination host. Thus, MMS_S is only an upper bound on the datagram size that a transport protocol may send; even when MMS_S exceeds 556, the transport layer must limit its messages to 556 bytes in the absence of other knowledge about the destination host.

(b) Some transport protocols (e.g., TCP) provide a way to explicitly inform the sender about the largest datagram the other end can receive and reassemble "The TCP Maximum Segment Size Option and Related Topics" [RFC-879]. There is no corresponding mechanism in the IP layer.

A transport protocol that assumes an EMTU_R larger than 576 (see Reassembly), can send a datagram of this larger size to another host that implements the same protocol.

(c) Hosts should ideally limit their EMTU_S for a given destination to the minimum MTU of all the networks along the path, to avoid any fragmentation. IP fragmentation, while formally correct, can create a serious transport protocol performance problem, because loss of a single fragment means all the fragments in the segment must be retransmitted [IP:9].

Since nearly all networks in the Internet currently support an MTU of 576 or greater, we strongly recommend the use of 576 for datagrams sent to non-local networks.

It has been suggested that a host could determine the MTU over a given path by sending a zero-offset datagram fragment and waiting for the receiver to time out the reassembly (which cannot complete!) and return an ICMP Time Exceeded message. This message would include the largest remaining fragment header in its body. More direct mechanisms are being experimented with, but have not yet been adopted (see e.g., RFC-1063).

RFC-791 Internet Protocol - Fragmentation Procedure

Example Implementation

The maximum sized datagram that can be transmitted through the next network is called the maximum transmission unit (MTU).

If the total length is less than or equal the maximum transmission unit then submit this datagram to the next step in datagram processing; otherwise cut the datagram into two fragments, the first fragment being the maximum size, and the second fragment being the rest of the datagram. The first fragment is submitted to the next step in datagram processing, while the second fragment is submitted to this procedure in case it is still too large.

Notation:

FO	Fragment Offset
IHL	Internet Header Length
DF	Don't Fragment flag
MF	More Fragments flag
TL	Total Length
OFO	Old Fragment Offset
OIHL	Old Internet Header Length
OMF	Old More Fragments flag
OTL	Old Total Length
NFB	Number of Fragment Blocks
MTU	Maximum Transmission Unit

Procedure:

IF TL =< MTU THEN Submit this datagram to the next step in datagram processing ELSE IF DF 1 THEN discard the datagram ELSE

To produce the first fragment:

(1) Copy the original internet header;

(2) OIHL <- IHL; OTL <- TL; OFO <- FO; OMF <- MF;

(3) NFB <- (MTU-IHL*4)/8;

(4) Attach the first NFB*8 data octets;

(5) Correct the header:
MF <- 1; TL <- (IHL*4)+(NFB*8);
Recompute Checksum;

(6) Submit this fragment to the next step in datagram processing; To produce the second fragment:

(7) Selectively copy the internet header (some options are not copied, see option definitions);

(8) Append the remaining data;

(9) Correct the header:
IHL <- (((OIHL*4)-(length of options not copied))+3)/4;
TL <- OTL - NFB*8 - (OIHL-IHL)*4);

FO <- OFO + NFB; MF <- OMF; Recompute Checksum;

(10) Submit this fragment to the fragmentation test; DONE.

In the above procedure each fragment (except the last) was made the maximum allowable size. An alternative might produce less than the maximum size datagrams. For example, one could implement a fragmentation procedure that repeatly divided large datagrams in half until the resulting fragments were less than the maximum transmission unit size.

Reassembly Procedure

RFC-791 Internet Protocol - Reassembly Procedure

Overview

The IP layer **must** implement reassembly of IP datagrams.

We designate the largest datagram size that can be reassembled by EMTU_R ("Effective MTU to receive"); this is sometimes called the "reassembly buffer size". EMTU_R **must** be greater than or equal to 576, **should** be either configurable or indefinite, and **should** be greater than or equal to the MTU of the connected network(s).

> **Discussion**
> A fixed EMTU_R limit should not be built into the code because some application layer protocols require EMTU_R values larger than 576.
>
> **Implementation**
> An implementation may use a contiguous reassembly buffer for each datagram, or it may use a more complex data structure that places no definite limit on the reassembled datagram size; in the latter case, EMTU_R is said to be "indefinite".
>
> Logically, reassembly is performed by simply copying each fragment into the packet buffer at the proper offset. Note that fragments may overlap if successive retransmissions use different packetizing but the same reassembly Id.
>
> The tricky part of reassembly is the bookkeeping to determine when all bytes of the datagram have been reassembled. We recommend Clark's algorithm as described in RFC-815 "IP Datagram Reassembly Algorithms" that requires no additional data space for the bookkeeping. However, note that, contrary to what is described in that document the first fragment header needs to be saved for inclusion in a possible ICMP Time Exceeded (Reassembly Timeout) message.

There **must** be a mechanism by which the transport layer can learn MMS_R, the maximum message size that can be received and reassembled in an IP datagram (see GET_MAXSIZES calls in Section 3.4). If EMTU_R is not indefinite, then the value of MMS_R is given by:

> MMS_R = EMTU_R - 20

since 20 is the minimum size of an IP header.

There **must** be a reassembly timeout. The reassembly timeout value **should** be a fixed value, not set from the remaining TTL. It is recommended that the value lie between 60 seconds and 120 seconds. If this timeout expires, the partially-reassembled datagram **must** be discarded and an ICMP Time Exceeded message sent to the source host (if fragment zero has been received).

RFC-791 Internet Protocol - Reassembly Procedure

Description

For each datagram the buffer identifier is computed as the concatenation of the source, destination, protocol, and identification fields. If this is a whole datagram (that is both the fragment offset and the more fragments fields are zero), then any reassembly resources associated with this buffer identifier are released and the datagram is forwarded to the next step in datagram processing.

If no other fragment with this buffer identifier is on hand then reassembly resources are allocated. The reassembly resources consist of a data buffer, a header buffer, a fragment block bit table, a total data length field, and a timer. The data from the fragment is placed in the data buffer according to its fragment offset and length, and bits are set in the fragment block bit table corresponding to the fragment blocks received.

If this is the first fragment (that is the fragment offset is zero) this header is placed in the header buffer. If this is the last fragment (that is the more fragments field is zero) the total data length is computed. If this fragment completes the datagram (tested by checking the bits set in the fragment block table), then the datagram is sent to the next step in datagram processing.

If the timer runs out, the all reassembly resources for this buffer identifier are released. The initial setting of the timer is a lower bound on the reassembly waiting time. This is because the waiting time will be increased if the Time to Live in the arriving fragment is greater than the current timer value but will not be decreased if it is less. The maximum this timer value could reach is the maximum time to live (approximately 4.25 minutes). The current recommendation for the initial timer setting is 60 - 120 seconds. Note that the choice of this parameter value is related to the buffer capacity available and the data rate of the transmission medium; that is, data rate times timer value equals buffer size (e.g., 10Kb/s X 15s = 150Kb).

Timer

The IP specification says that the reassembly timeout should be the remaining TTL from the IP header, but this does not work well because gateways generally treat TTL as a simple hop count rather than an elapsed time. If the reassembly timeout is too small, datagrams will be discarded unnecessarily, and communication may fail. The timeout needs to be at least as large as the typical maximum delay across the Internet. A realistic minimum reassembly timeout would be 60 seconds.

It has been suggested that a cache might be kept of round-trip times measured by transport protocols for various destinations, and that these values might be used to dynamically determine a reasonable reassembly timeout value. Further investigation of this approach is required.

If the reassembly timeout is set too high, buffer resources in the receiving host will be tied up too long, and the MSL (Maximum Segment Lifetime) [TCP:1] will be larger than necessary. The MSL controls the maximum rate at which fragmented datagrams can be sent using distinct values of the 16-bit Ident field; a larger MSL lowers the maximum rate. The TCP specification [TCP:1] arbitrarily assumes a value of 2 minutes for MSL. This sets an upper limit on a reasonable reassembly timeout value.

RFC-791 Internet Protocol - Reassembly Procedure

Example Implementation

Notation:

FO	Fragment Offset
IHL	Internet Header Length
MF	More Fragments flag
TTL	Time To Live
NFB	Number of Fragment Blocks
TL	Total Length
TDL	Total Data Length
BUFID	Buffer Identifier
RCVBT	Fragment Received Bit Table
TLB	Timer Lower Bound

Procedure:

(1) BUFID <- source|destination|protocol|identification;

(2) IF FO = 0 AND MF = 0

(3) THEN IF buffer with BUFID is allocated

(4) THEN flush all reassembly for this BUFID;

(5) Submit datagram to next step; DONE.

(6) ELSE IF no buffer with BUFID is allocated

(7) THEN allocate reassembly resources with BUFID;
TIMER <- TLB; TDL <- 0;

(8) put data from fragment into data buffer with BUFID from octet
FO*8 to octet (TL-IHL*4))+FO*8;

(9) set RCVBT bits from FO to FO+((TL-(IHL*4)+7)/8);

(10) IF MF = 0 THEN TDL <- TL-(IHL*4)+(FO*8)

(11) IF FO = 0 THEN put header in header buffer

(12) IF TDL # 0

(13) AND all RCVBT bits from 0 to (TDL+7)/8 are set

(14) THEN TL <- TDL+(IHL*4)

(15) Submit datagram to next step;

(16) free all reassembly resources for this BUFID;
DONE.

(17) TIMER <- MAX(TIMER,TTL);

(18) give up until next fragment or timer expires;

(19) timer expires: flush all reassembly with this BUFID; DONE.

In the case that two or more fragments contain the same data either identically or through a partial overlap, this procedure will use the more recently arrived copy in the data buffer and datagram delivered.

Identification

The choice of the Identifier for a datagram is based on the need to provide a way to uniquely identify the fragments of a particular datagram. The protocol module assembling fragments judges fragments to belong to the same datagram if they have the same source, destination, protocol, and Identifier. Thus, the sender must choose the Identifier to be unique for this source, destination pair and protocol for the time the datagram (or any fragment of it) could be alive in the internet.

It seems then that a sending protocol module needs to keep a table of Identifiers, one entry for each destination it has communicated with in the last maximum packet lifetime for the internet.

However, since the Identifier field allows 65,536 different values, some host may be able to simply use unique identifiers independent of destination.

It is appropriate for some higher level protocols to choose the identifier. For example, TCP protocol modules may retransmit an identical TCP segment, and the probability for correct reception would be enhanced if the retransmission carried the same identifier as the original transmission since fragments of either datagram could be used to construct a correct TCP segment.

Discussion

Some Internet protocol experts have maintained that when a host sends an identical copy of an earlier datagram, the new copy should contain the same Identification value as the original. There are two suggested advantages: (1) if the datagrams are fragmented and some of the fragments are lost, the receiver may be able to reconstruct a complete datagram from fragments of the original and the copies; (2) a congested gateway might use the IP Identification field (and Fragment Offset) to discard duplicate datagrams from the queue.

However, the observed patterns of datagram loss in the Internet do not favor the probability of retransmitted fragments filling reassembly gaps, while other mechanisms (e.g., TCP repacketizing upon retransmission) tend to prevent retransmission of an identical datagram [IP:9]. Therefore, we believe that retransmitting the same Identification field is not useful. Also, a connectionless transport protocol like UDP would require the cooperation of the application programs to retain the same Identification value in identical datagrams.

Type of Service

The type of service (TOS) is for internet service quality selection. The type of service is specified along the abstract parameters precedence, delay, throughput, and reliability. These abstract parameters are to be mapped into the actual service parameters of the particular networks the datagram traverses.

The "Type-of-Service" byte in the IP header is divided into two sections: the Precedence field (high-order 3 bits), and a field that is customarily called "Type-of-Service" or "TOS" (low-order 5 bits). In this document, all references to "TOS" or the "TOS field" refer to the low-order 5 bits only.

Precedence An independent measure of the importance of this datagram. This field is intended for Department of Defense applications of the Internet Protocols. Please consult the Defense Communications Agency (DCA) for guidance on the use of this field. Vendors should note that the use of precedence will likely require that its value be passed between protocol layers just as the TOS field is.

Delay Prompt delivery is important for datagrams with this indication.

Throughput High data rate is important for datagrams with this indication.

Reliability A higher level of effort to ensure delivery is important for datagrams with this indication.

For example, the ARPANET has a priority bit, and a choice between "standard" messages (type 0) and "uncontrolled" messages (type 3), (the choice between single packet and multipacket messages can also be considered a service parameter). The uncontrolled messages tend to be less reliably delivered and suffer less delay. Suppose an internet datagram is to be sent through the ARPANET. Let the internet type of service be given as:

Precedence: 5
Delay: 0
Throughput: 1
Reliability: 1

In this example, the mapping of these parameters to those available for the ARPANET would be to set the ARPANET priority bit on since the Internet precedence is in the upper half of its range, to select standard messages since the throughput and reliability requirements are indicated and delay is not. More details are given on service mappings in "Service Mappings".

The IP layer **must** provide a means for the transport layer to set the TOS field of every datagram that is sent; the default is all zero bits. The IP layer **should** pass received TOS values up to the transport layer.

The particular link-layer mappings of TOS contained in RFC- 795 **should not** be implemented.

Discussion
While the TOS field has been little used in the past, it is expected to play an increasing role in the near future. The TOS field is expected to be used to control two aspects of gateway operations: routing and queueing algorithms. See **Type of Service** from "Requirements for Internet Hosts -- Application and Support" [RFC-1123] for the requirements on application programs to specify TOS values.

The TOS field may also be mapped into link-layer service selectors. This has

been applied to provide effective sharing of serial lines by different classes of TCP traffic, for example. However, the mappings suggested in "Service Mappings" [RFC-795] for networks that were included in the Internet as of 1981 are now obsolete.

RFC-791 Internet Protocol - Discussion

Time to Live

The time to live is set by the sender to the maximum time the datagram is allowed to be in the internet system. If the datagram is in the internet system longer than the time to live, then the datagram must be destroyed.

This field must be decreased at each point that the internet header is processed to reflect the time spent processing the datagram. Even if no local information is available on the time actually spent, the field must be decremented by 1. The time is measured in units of seconds (i.e. the value 1 means one second). Thus, the maximum time to live is 255 seconds or 4.25 minutes. Since every module that processes a datagram must decrease the TTL by at least one even if it process the datagram in less than a second, the TTL must be thought of only as an upper bound on the time a datagram may exist. The intention is to cause undeliverable datagrams to be discarded, and to bound the maximum datagram lifetime.

Some higher level reliable connection protocols are based on assumptions that old duplicate datagrams will not arrive after a certain time elapses. The TTL is a way for such protocols to have an assurance that their assumption is met.

A host **must not** send a datagram with a Time-to-Live (TTL) value of zero.

A host **must not** discard a datagram just because it was received with TTL less than 2.

The IP layer **must** provide a means for the transport layer to set the TTL field of every datagram that is sent. When a fixed TTL value is used, it **must** be configurable. The suggested value for Time to Live is published in Assigned Numbers, as of March 1990 this value is 32.

Discussion

The TTL field has two functions: limit the lifetime of TCP segments, and terminate Internet routing loops. Although TTL is a time in seconds, it also has some attributes of a hop- count, since each gateway is required to reduce the TTL field by at least one.

The intent is that TTL expiration will cause a datagram to be discarded by a gateway but not by the destination host; however, hosts that act as gateways by forwarding datagrams must follow the gateway rules for TTL.

A higher-layer protocol may want to set the TTL in order to implement an "expanding scope" search for some Internet resource. This is used by some diagnostic tools, and is expected to be useful for locating the "nearest" server of a given class using IP multicasting, for example. A particular transport protocol may also want to specify its own TTL bound on maximum datagram lifetime.

A fixed value must be at least big enough for the Internet "diameter," i.e., the longest possible path. A reasonable value is about twice the diameter, to allow for continued Internet growth.

Options

The options are optional in each datagram, but required in implementations. That is, the presence or absence of an option is the choice of the sender, but each internet module must be able to parse every option. There can be several options present in the option field.

The options might not end on a 32-bit boundary. The internet header must be filled out with octets of zeros. The first of these would be interpreted as the end-of-options option, and the remainder as internet header padding.

Every internet module must be able to act on every option. The Security Option is required if classified, restricted, or compartmented traffic is to be passed.

There **must** be a means for the transport layer to specify IP options to be included in transmitted IP datagrams (See Internet/Transport Layer Interface).

All IP options (except NOP or END-OF-LIST) received in datagrams **must** be passed to the transport layer (or to ICMP processing when the datagram is an ICMP message). The IP and transport layer **must** each interpret those IP options that they understand and silently ignore the others.

Later sections of this document discuss specific IP option support required by each of ICMP, TCP, and UDP. (See Implementation).

> **Discussion**
> Passing all received IP options to the transport layer is a deliberate "violation of strict layering" that is designed to ease the introduction of new transport-relevant IP options in the future. Each layer must pick out any options that are relevant to its own processing and ignore the rest. For this purpose, every IP option except NOP and END-OF-LIST will include a specification of its own length.
>
> This document does not define the order in which a receiver must process multiple options in the same IP header. Hosts sending multiple options must be aware that this introduces an ambiguity in the meaning of certain options when combined with a source-route option.

Implementation

The IP layer must not crash as the result of an option length that is outside the possible range. For example, erroneous option lengths have been observed to put some IP implementations into infinite loops.

Here are the requirements for specific IP options:

Security Option
Stream Identifier
Source Route Options
Record Route Option
Timestamp Option

Security Option

Some environments require the Security option in every datagram; such a requirement is outside the scope of this document and the IP standard specification. Note, however, that the security options described in this RFC (791) and RFC-1038 are obsolete. For DoD applications, vendors should consult "Internet Protocol Security Options" [RFC-1108] for guidance.

Stream Identifier Option

This option is obsolete; it **should not** be sent, and it **must** be silently ignored if received.

Source Route Options

A host **must** support originating a source route and **must** be able to act as the final destination of a source route.

If host receives a datagram containing a completed source route (i.e., the pointer points beyond the last field), the datagram has reached its final destination; the option as received (the recorded route) **must** be passed up to the transport layer (or to ICMP message processing). This recorded route will be reversed and used to form a return source route for reply datagrams (see discussion of IP Options in Section 4). When a return source route is built, it **must** be correctly formed even if the recorded route included the source host (see case (B) in the discussion below).

An IP header containing more than one Source Route option **must not** be sent; the effect on routing of multiple Source Route options is implementation-specific.

Source Route Forwarding presents the rules for a host acting as an intermediate hop in a source route, i.e., forwarding a source-routed datagram. (See Discussion).

Record Route Option

Implementation of originating and processing the Record Route option is **optional**.

Timestamp Option

Implementation of originating and processing the Timestamp option is OPTIONAL. If it is implemented, the following rules apply:

- The originating host **must** record a timestamp in a Timestamp option whose Internet address fields are not pre-specified or whose first pre-specified address is the host's interface address.
- The destination host **must** (if possible) add the current timestamp to a Timestamp option before passing the option to the transport layer or to ICMP for processing.
- A timestamp value **must** follow the rules given in ICMP Timestamp for the ICMP Timestamp message.

Source Route Option Discussion

If a source-routed datagram is fragmented, each fragment will contain a copy of the source route. Since the processing of IP options (including a source route) must precede reassembly, the original datagram will not be reassembled until the final destination is reached.

Suppose a source routed datagram is to be routed from host S to host D via gateways G1, G2, ... Gn. There was an ambiguity in the specification over whether the source route option in a datagram sent out by S should be (A) or (B):

(A): {>>G2, G3, ... Gn, D} <--- CORRECT

(B): {S, >>G2, G3, ... Gn, D} <---- WRONG

(where >> represents the pointer). If (A) is sent, the datagram received at D will contain the option: {G1, G2, ... Gn >>}, with S and D as the IP source and destination addresses. If (B) were sent, the datagram received at D would again contain S and D as the same IP source and destination addresses, but the option would be: {S, G1, ...Gn >>}; i.e., the originating host would be the first hop in the route.

Checksum

The internet header checksum is recomputed if the internet header is changed. For example, a reduction of the time to live, additions or changes to internet options, or due to fragmentation. This checksum at the internet level is intended to protect the internet header fields from transmission errors.

There are some applications where a few data bit errors are acceptable while retransmission delays are not. If the internet protocol enforced data correctness such applications could not be supported.

A host **must** verify the IP header checksum on every received datagram and silently dicsard every datagram that has a bad checksum.

Errors

Internet protocol errors may be reported via the ICMP messages.

Interfaces

The functional description of user interfaces to the IP is, at best, fictional, since every operating system will have different facilities. Consequently, we must warn readers that different IP implementations may have different user interfaces. However, all IPs must provide a certain minimum set of services to guarantee that all IP implementations can support the same protocol hierarchy. This section specifies the functional interfaces required of all IP implementations.

Internet protocol interfaces on one side to the local network and on the other side to either a higher level protocol or an application program. In the following, the higher level protocol or application program (or even a gateway program) will be called the "user" since it is using the internet module. Since internet protocol is a datagram protocol, there is minimal memory or state maintained between datagram transmissions, and each call on the internet protocol module by the user supplies all information necessary for the IP to perform the service requested.

An Example Upper Level Interface

The following two example calls satisfy the requirements for the user to internet protocol module communication ("=>" means returns):

SEND (src, dst, prot, TOS, TTL, BufPTR, len, Id, DF, opt => result)

where:

src = source address
dst = destination address
prot = protocol
TOS = type of service
TTL = time to live
BufPTR = buffer pointer
len = length of buffer
Id = Identifier
DF = Don't Fragment
opt = option data
result = response
OK = datagram sent ok
Error = error in arguments or local network error

Note that the precedence is included in the TOS and the security/compartment is passed as an option.

RECV (BufPTR, prot, => result, src, dst, TOS, len, opt)

where:

BufPTR = buffer pointer
prot = protocol
result = response
OK = datagram received ok
Error = error in arguments
len = length of buffer
src = source address
dst = destination address
TOS = type of service
opt = option data

When the user sends a datagram, it executes the SEND call supplying all the arguments. The internet protocol module, on receiving this call, checks the arguments and prepares and sends the message. If the arguments are good and the datagram is accepted by the local network, the call returns successfully. If either the arguments are bad, or the datagram is not accepted by the local network, the call returns unsuccessfully. On unsuccessful returns, a reasonable report must be made as to the cause of the problem, but the details of such reports are up to individual implementations.

When a datagram arrives at the internet protocol module from the local network, either there is a pending RECV call from the user addressed or there is not. In the first case, the pending call is satisfied by passing the information from the datagram to the user. In the second case, the user addressed is notified of a pending datagram. If the user addressed

does not exist, an ICMP error message is returned to the sender, and the data is discarded.

The notification of a user may be via a pseudo interrupt or similar mechanism, as appropriate in the particular operating system environment of the implementation.

A user's RECV call may then either be immediately satisfied by a pending datagram, or the call may be pending until a datagram arrives.

The source address is included in the send call in case the sending host has several addresses (multiple physical connections or logical addresses). The internet module must check to see that the source address is one of the legal address for this host.

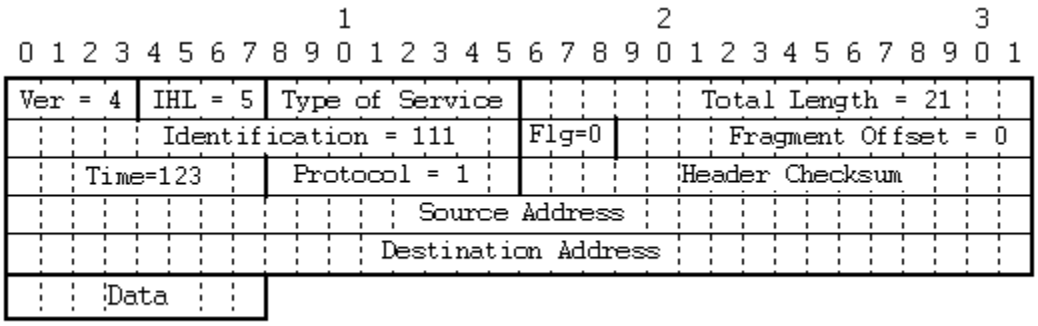
An implementation may also allow or require a call to the internet module to indicate interest in or reserve exclusive use of a class of datagrams (e.g., all those with a certain value in the protocol field).

This section functionally characterizes a USER/IP interface. The notation used is similar to most procedure of function calls in high level languages, but this usage is not meant to rule out trap type service calls (e.g., SVCs, UUOs, EMTs), or any other form of interprocess communication.

RFC-791 Internet Protocol - Examples

Example 1

This is an example of the minimal data carrying internet datagram:



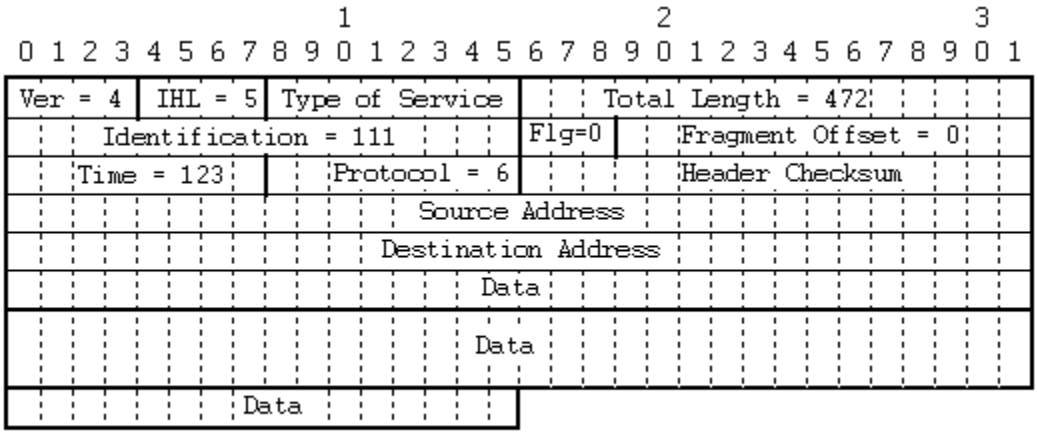
Example Internet Datagram

This is a internet datagram in version 4 of internet protocol; the internet header consists of five 32 bit words, and the total length of the datagram is 21 octets. This datagram is a complete datagram (not a fragment).

RFC-791 Internet Protocol - Examples

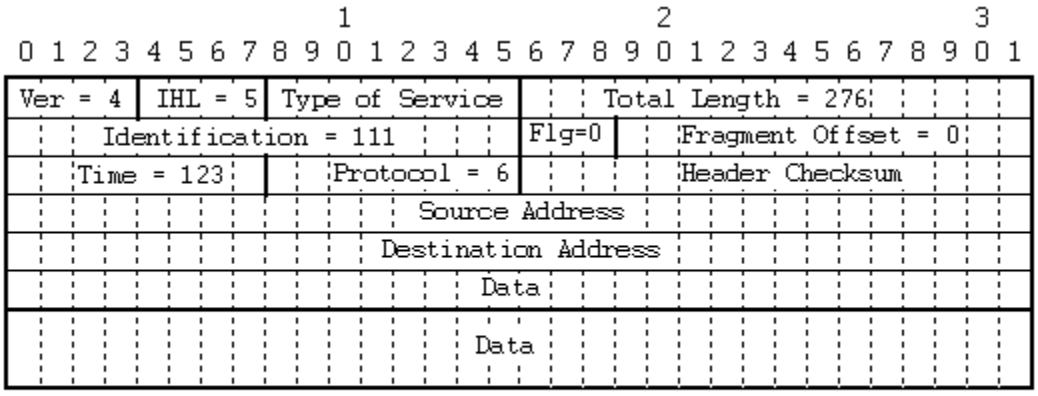
Example 2

In this example, we show first a moderate size internet datagram (452 data octets), then two internet fragments that might result from the fragmentation of this datagram if the maximum sized transmission allowed were 280 octets.



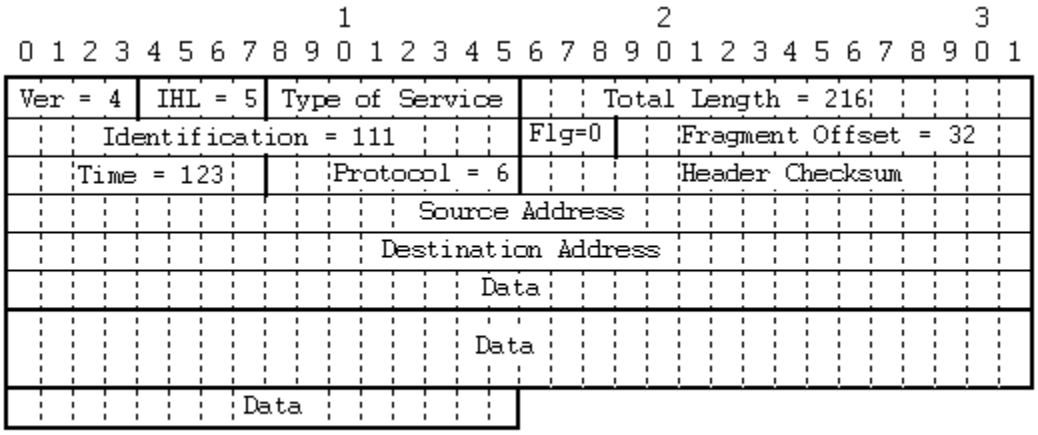
Example Internet Datagram

Now the first fragment that results from splitting the datagram after 256 data octets.



Example Internet Datagram

And the second fragment.

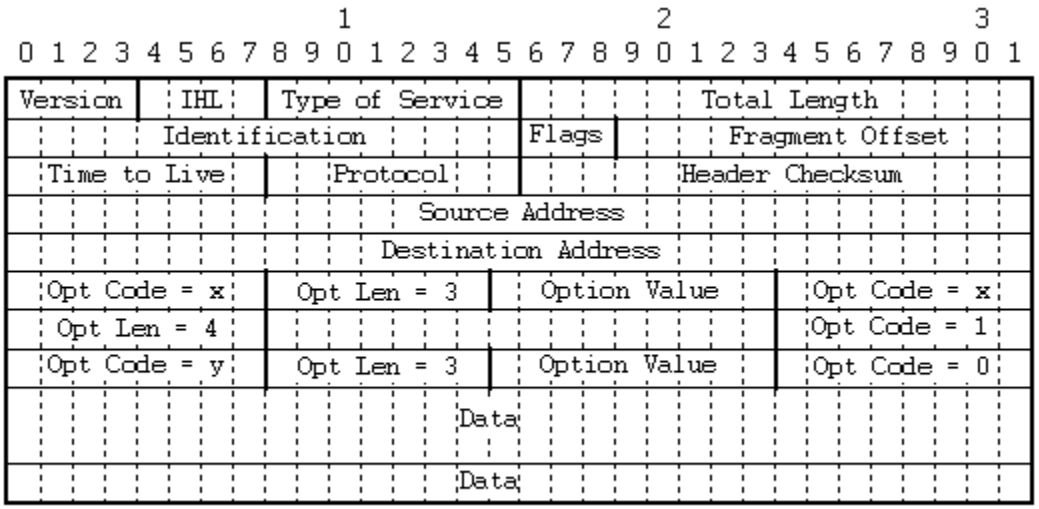


Example Internet Datagram

RFC-791 Internet Protocol - Examples

Example 3

Here, we show an example of a datagram containing options:

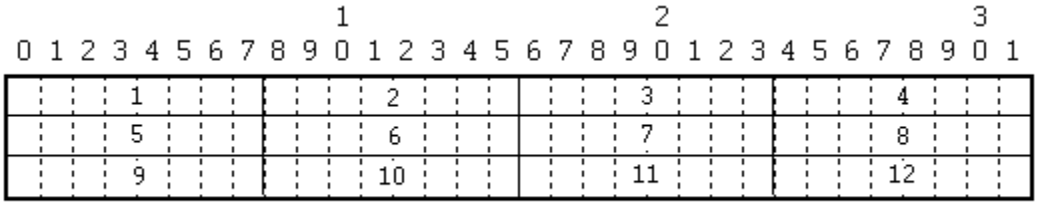


Example Internet Datagram

RFC-791 Internet Protocol

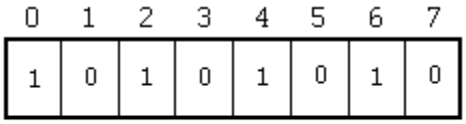
Data Transmission Order

The order of transmission of the header and data described in this document is resolved to the octet level. Whenever a diagram shows a group of octets, the order of transmission of those octets is the normal order in which they are read in English. For example, in the following diagram the octets are transmitted in the order they are numbered.



Transmission Order of Bytes

Whenever an octet represents a numeric quantity the left most bit in the diagram is the high order or most significant bit. That is, the bit labeled 0 is the most significant bit. For example, the following diagram represents the value 170 (decimal).



Significance of Bits

Decimal Value = 170

Similarly, whenever a multi-octet field represents a numeric quantity the left most bit of the whole field is the most significant bit. When a multi-octet quantity is transmitted the most significant octet is transmitted first.

Glossary

1822
BBN Report 1822, "The Specification of the Interconnection of a Host and an IMP". The specification of interface between a host and the ARPANET.

ARPANET leader
The control information on an ARPANET message at the host-IMP interface.

ARPANET message
The unit of transmission between a host and an IMP in the ARPANET. The maximum size is about 1012 octets (8096 bits).

ARPANET packet
A unit of transmission used internally in the ARPANET between IMPs. The maximum size is about 126 octets (1008 bits).

Destination
The destination address, an internet header field.

DF
The Don't Fragment bit carried in the flags field.

Flags
An internet header field carrying various control flags.

Fragment Offset
This internet header field indicates where in the internet datagram a fragment belongs.

GGP
Gateway to Gateway Protocol, the protocol used primarily between gateways to control routing and other gateway functions.

header
Control information at the beginning of a message, segment, datagram, packet or block of data.

ICMP
Internet Control Message Protocol, implemented in the internet module, the ICMP is used from gateways to hosts and between hosts to report errors and make routing suggestions.

Identification
An internet header field carrying the identifying value assigned by the sender to aid in assembling the fragments of a datagram.

IHL
The internet header field Internet Header Length is the length of the internet header measured in 32 bit words.

IMP
The Interface Message Processor, the packet switch of the ARPANET.

Internet Address

A four octet (32 bit) source or destination address consisting of a Network field and a Local Address field.

internet datagram
The unit of data exchanged between a pair of internet modules (includes the internet header).

internet fragment
A portion of the data of an internet datagram with an internet header.

Local Address
The address of a host within a network. The actual mapping of an internet local address on to the host addresses in a network is quite general, allowing for many to one mappings.

MF
The More-Fragments Flag carried in the internet header flags field.

module
An implementation, usually in software, of a protocol or other procedure.

more-fragments flag
A flag indicating whether or not this internet datagram contains the end of an internet datagram, carried in the internet header Flags field.

NFB
The Number of Fragment Blocks in a the data portion of an internet fragment. That is, the length of a portion of data measured in 8 octet units.

octet
An eight bit byte.

Options
The internet header Options field may contain several options, and each option may be several octets in length.

Padding
The internet header Padding field is used to ensure that the data begins on 32 bit word boundary. The padding is zero.

Protocol
In this document, the next higher level protocol identifier, an internet header field.

Rest
The local address portion of an Internet Address.

Source
The source address, an internet header field.

TCP
Transmission Control Protocol: A host-to-host protocol for reliable communication in internet environments.

TCP Segment
The unit of data exchanged between TCP modules (including the TCP header).

TFTP
Trivial File Transfer Protocol: A simple file transfer protocol built on UDP.

Time to Live
An internet header field which indicates the upper bound on how long this internet datagram may exist.

TOS
Type of Service

Total Length
The internet header field Total Length is the length of the datagram in octets including internet header and data.

TTL
Time to Live

Type of Service
An internet header field which indicates the type (or quality) of service for this internet datagram.

UDP
User Datagram Protocol: A user level protocol for transaction oriented applications.

User
The user of the internet protocol. This may be a higher level protocol module, an application program, or a gateway program.

Version
The Version field indicates the format of the internet header.

4.2BSD and its derivatives, but not 4.3BSD.

RFC-792 Internet Control Message Protocol

Jon Postel
USC/Information Sciences Institute
September 1981

DARPA Internet Program
Protocol Specification

This section includes the original text of RFC-792 as well as many corrections, comments, and suggestions made since its publication. There are also links to other relevant documents and discussions.

RFC-792 Internet Control Message Protocol

Introduction

The Internet Protocol (IP) [RFC-791] is used for host-to-host datagram service in a system of interconnected networks called the Catenet [2]. The network connecting devices are called Gateways. These gateways communicate between themselves for control purposes via a Gateway to Gateway Protocol (GGP) [3,4]. Occasionally a gateway or destination host will communicate with a source host, for example, to report an error in datagram processing. For such purposes this protocol, the Internet Control Message Protocol (ICMP), is used. ICMP, uses the basic support of IP as if it were a higher level protocol, however, ICMP is actually an integral part of IP, and must be implemented by every IP module.

ICMP messages are sent in several situations: for example, when a datagram cannot reach its destination, when the gateway does not have the buffering capacity to forward a datagram, and when the gateway can direct the host to send traffic on a shorter route.

The Internet Protocol is not designed to be absolutely reliable. The purpose of these control messages is to provide feedback about problems in the communication environment, not to make IP reliable. There are still no guarantees that a datagram will be delivered or a control message will be returned. Some datagrams may still be undelivered without any report of their loss. The higher level protocols that use IP must implement their own reliability procedures if reliable communication is required.

The ICMP messages typically report errors in the processing of datagrams. To avoid the infinite regress of messages about messages etc., no ICMP messages are sent about ICMP messages. Also ICMP messages are only sent about errors in handling fragment zero of fragemented datagrams. (Fragment zero has the fragment offeset equal zero).

RFC-792 Internet Control Message Protocol

Message Formats

ICMP messages are sent using the basic IP header. The first octet of the data portion of the datagram is a ICMP type field; the value of this field determines the format of the remaining data. Any field labeled "unused" is reserved for later extensions and must be zero when sent, but receivers should not use these fields (except to include them in the checksum). Unless otherwise noted under the individual format descriptions, the values of the internet header fields are as follows:

Version

4

IHL

Internet header length in 32-bit words.

Type of Service

0

Total Length

Length of internet header and data in octets.

Identification, Flags, Fragment Offset

Used in fragmentation.

Time to Live

Time to live in seconds; as this field is decremented at each machine in which the datagram is processed, the value in this field should be at least as great as the number of gateways which this datagram will traverse.

Protocol

ICMP = 1

Header Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum , the checksum field should be zero. This checksum may be replaced in the future.

Source Address

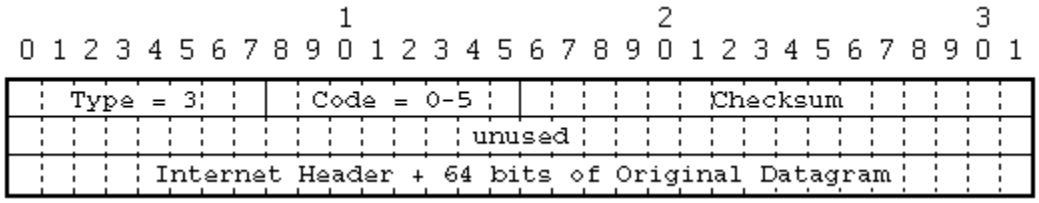
The address of the gateway or host that composes the ICMP message. Unless otherwise noted, this can be any of a gateway's addresses.

Destination Address

The address of the gateway or host to which the message should be sent.

RFC-792 ICMP - Message Formats

Destination Unreachable Message



IP Fields:

Destination Address

The source network and address from the original datagram's data.

ICMP Fields:

Type

3

Code

0 = net unreachable
1 = host unreachable
2 = protocol unreachable
3 = port unreachable
4 = fragmentation needed and DF set
5 = source route failed
6= destination network unknown
7 = destination host unknown
8 = source host isolated
9 = communication with destination network administratively prohibited
10 = communication with destination host administratively prohibited
11 = network unreachable for type of service
12 = host unreachable for type of service

Checksum

Internet Header + 64 bits of Data Datagram

Description

If, according to the information in the gateway's routing tables, the network specified in the internet destination field of a datagram is unreachable, e.g., the distance to the network is infinity, the gateway may send a destination unreachable message to the internet source host of the datagram. In addition, in some networks, the gateway may be able to determine if the internet destination host is unreachable. Gateways in these networks may send destination unreachable messages to the source host when the destination host is unreachable.

If, in the destination host, the IP module cannot deliver the datagram because the indicated protocol module or process port is not active, the destination host may send a destination unreachable message to the source host.

Another case is when a datagram must be fragmented to be forwarded by a

gateway yet the Don't Fragment flag is on. In this case the gateway must discard the datagram and may return a destination unreachable message.

Codes 0, 1, 4, and 5 may be received from a gateway.

A host **should** generate Destination Unreachable messages with code:

2 (Protocol Unreachable), when the designated transport protocol is not supported; or

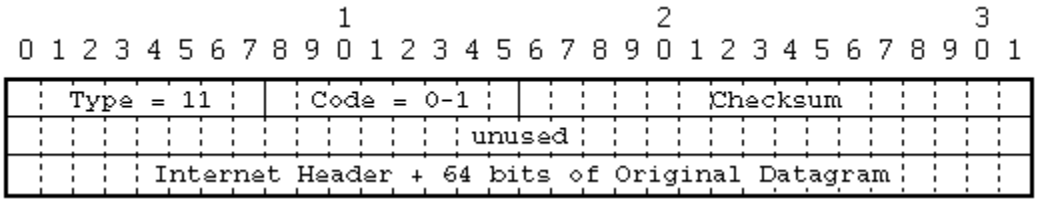
3 (Port Unreachable), when the designated transport protocol (e.g., UDP) is unable to demultiplex the datagram but has no protocol mechanism to inform the sender.

A Destination Unreachable message that is received **must** be reported to the transport layer. The transport layer **should** use the information appropriately; for example, see UDP/IP Options, TCP/ICMP, and TCP/Application Layer Interface. A transport protocol that has its own mechanism for notifying the sender that a port is unreachable (e.g., TCP, which sends RST segments) **must** nevertheless accept an ICMP Port Unreachable for the same purpose.

A Destination Unreachable message that is received with code **0** (Net), **1** (Host), or **5** (Bad Source Route) may result from a routing transient and **must** therefore be interpreted as only a hint, not proof, that the specified destination is unreachable (see "Fault Isolation and Recovery [RFC-816]. For example, it **must not** be used as proof of a dead gateway (see Routing Outbound Datagrams).

RFC-792 ICMP - Message Formats

Time Exceeded Message



IP Fields:

Destination Address

The source network and address from the original datagram's data.

ICMP Fields:

Type

11

Code

0 = time to live exceeded in transit;
1 = fragment reassembly time exceeded.

Checksum

Internet Header + 64 bits of Data Datagram

Description

If the gateway processing a datagram finds the time to live field is zero it must discard the datagram. The gateway may also notify the source host via the time exceeded message.

If a host reassembling a fragmented datagram cannot complete the reassembly due to missing fragments within its time limit it discards the datagram, and it may send a time exceeded message. In the future, receipt of this message might be part of some "MTU discovery" procedure, to discover the maximum datagram size that can be sent on the path without fragmentation.

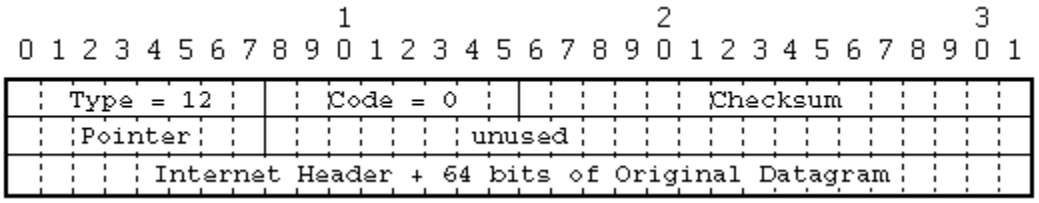
If fragment zero is not available then no time exceeded need be sent at all.

Code 0 **may** be received from a gateway. Code 1 **may** be received from a host.

An incoming Time Exceeded message **must** be passed to the transport layer.

A gateway will send a Time Exceeded Code 0 (In Transit) message when it discards a datagram due to an expired TTL field. This indicates either a gateway routing loop or too small an initial TTL value.

Parameter Problem Message



IP Fields:

Destination Address

The source network and address from the original datagram's data.

ICMP Fields:

Type

12

Code

0 = pointer indicates the error.
1 = required option is missing.

Checksum

Pointer

If code = 0, identifies the octet where an error was detected.

Internet Header + 64 bits of Data Datagram

Description

If the gateway or host processing a datagram finds a problem with the header parameters such that it cannot complete processing the datagram it must discard the datagram. One potential source of such a problem is with incorrect arguments in an option. The gateway or host may also notify the source host via the parameter problem message. This message is only sent if the error caused the datagram to be discarded.

A host **should** generate Parameter Problem messages. An incoming Parameter Problem message **must** be passed to the transport layer, and it **may** be reported to the user.

The ICMP Parameter Problem message is sent to the source host for any problem not specifically covered by another ICMP message. Receipt of a Parameter Problem message generally indicates some local or remote implementation error.

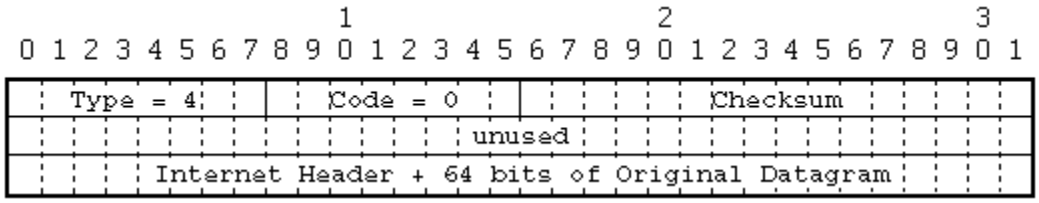
The pointer identifies the octet of the original datagram's header where the error was detected (it may be in the middle of an option). For example, 1 indicates something is wrong with the Type of Service, and (if there are options present) 20 indicates something is wrong with the type code of the first option.

Code 0 **may** be received from a gateway or a host.

Code 1 is currently in use in the military community for a missing security option.

RFC-792 ICMP - Message Formats

Source Quench Message



IP Fields:

Destination Address

The source network and address of the original datagram's data.

ICMP Fields:

Type

4

Code

0

Checksum

Internet Header + 64 bits of Data Datagram

Description

A gateway may discard internet datagrams if it does not have the buffer space needed to queue the datagrams for output to the next network on the route to the destination network. If a gateway discards a datagram, it may send a source quench message to the internet source host of the datagram. A destination host may also send a source quench message if datagrams arrive too fast to be processed. The source quench message is a request to the host to cut back the rate at which it is sending traffic to the internet destination. The gateway may send a source quench message for every message that it discards. On receipt of a source quench message, the source host should cut back the rate at which it is sending traffic to the specified destination until it no longer receives source quench messages from the gateway. The source host can then gradually increase the rate at which it sends traffic to the destination until it again receives source quench messages.

The gateway or host may send the source quench message when it approaches its capacity limit rather than waiting until the capacity is exceeded. This means that the data datagram which triggered the source quench message may be delivered.

Code 0 may be received from a gateway or a host.

A host **may** send a Source Quench message if it is approaching, or has reached, the point at which it is forced to discard incoming datagrams due to a shortage of reassembly buffers or other resources. See Section 2.2.3 of [INTRO:2] for suggestions on when to send Source Quench.

If a Source Quench message is received, the IP layer **must** report it to the transport layer (or ICMP processing). In general, the transport or application layer **should** implement a

mechanism to respond to Source Quench for any protocol that can send a sequence of datagrams to the same destination and which can reasonably be expected to maintain enough state information to make this feasible. (See Discussion).

RFC-792 Internet Control Message Protocol

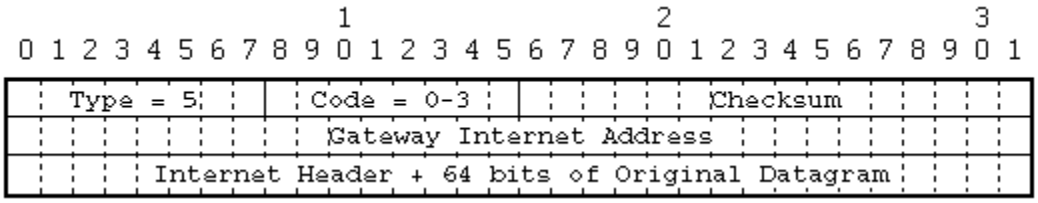
ICMP Source Quench Discussion

A Source Quench may be generated by the target host or by some gateway in the path of a datagram. The host receiving a Source Quench should throttle itself back for a period of time, then gradually increase the transmission rate again. The mechanism to respond to Source Quench may be in the transport layer (for connection-oriented protocols like TCP) or in the application layer (for protocols that are built on top of UDP).

A mechanism has been proposed, "Something a Host Could Do with Source Quench" [RFC-1016] to make the IP layer respond directly to Source Quench by controlling the rate at which datagrams are sent, however, this proposal is currently experimental and not currently recommended.

RFC-792 ICMP - Message Formats

Redirect Message



IP Fields:

Destination Address

The source network and address of the original datagram's data.

ICMP Fields:

Type

5

Code

0 = Redirect datagrams for the Network.
1 = Redirect datagrams for the Host.
2 = Redirect datagrams for the Type of Service and Network.
3 = Redirect datagrams for the Type of Service and Host.

Checksum

Gateway Internet Address

Address of the gateway to which traffic for the network specified in the internet destination network field of the original datagram's data should be sent.

Internet Header + 64 bits of Data Datagram

Description

The gateway sends a redirect message to a host in the following situation. A gateway, G1, receives an internet datagram from a host on a network to which the gateway is attached. The gateway, G1, checks its routing table and obtains the address of the next gateway, G2, on the route to the datagram's internet destination network, X. If G2 and the host identified by the internet source address of the datagram are on the same network, a redirect message is sent to the host. The redirect message advises the host to send its traffic for network X directly to gateway G2 as this is a shorter path to the destination. The gateway forwards the original datagram's data to its internet destination.

For datagrams with the IP source route options and the gateway address in the destination address field, a redirect message is not sent even if there is a better route to the ultimate destination than the next address in the source route.

Codes 0, 1, 2, and 3 may be received from a gateway.

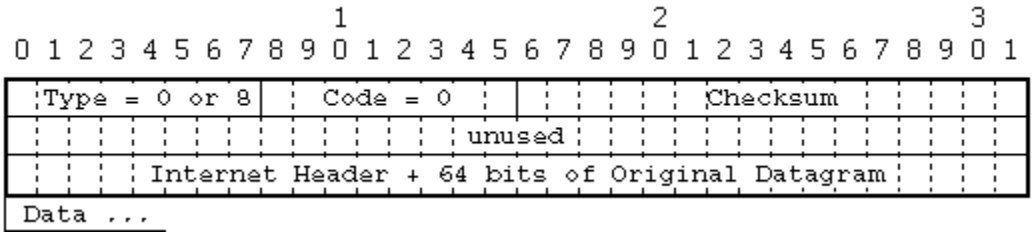
A host **should not** send an ICMP Redirect message; Redirects are to be sent only by gateways.

A host receiving a Redirect message **must** update its routing information accordingly. Every host **must** be prepared to accept both Host and Network Redirects and to process them as described in Gateway Selection.

A Redirect message **should** be silently discarded if the new gateway address it specifies is not on the same connected (sub-) net through which the Redirect arrived, or if the source of the Redirect is not the current first-hop gateway for the specified destination (see Routing Outbound Datagrams).

RFC-792 ICMP - Message Formats

Echo and Echo Reply Message



IP Fields:

Addresses

The address of the source in an echo message will be the destination of the echo reply message. To form an echo reply message, the source and destination addresses are simply reversed, the type code changed to 0, and the checksum recomputed.

ICMP Fields:

Type

8 for echo message;
0 for echo reply message.

Code

0

Checksum

Identifier

If code = 0, an identifier to aid in matching echos and replies, may be zero.

Sequence Number

If code = 0, a sequence number to aid in matching echos and replies, may be zero.

Description

The data received in the echo message must be returned in the echo reply message.

The identifier and sequence number may be used by the echo sender to aid in matching the replies with the echo requests. For example, the identifier might be used like a port in TCP or UDP to identify a session, and the sequence number might be incremented on each echo request sent. The echoer returns these same values in the echo reply.

Code 0 may be received from a gateway or a host.

Every host **must** implement an ICMP Echo server function that receives Echo Requests and sends corresponding Echo Replies. A host **should** also implement an application-layer interface for sending an Echo Request and receiving an Echo Reply, for diagnostic purposes.

An ICMP Echo Request destined to an IP broadcast or IP multicast address **may** be silently discarded. (See Discussion).

ICMP Echo Request/Reply Discussion

This neutral provision results from a passionate debate between those who feel that ICMP Echo to a broadcast address provides a valuable diagnostic capability and those who feel that misuse of this feature can too easily create packet storms.

The IP source address in an ICMP Echo Reply **must** be the same as the specific-destination address (defined in Addressing) of the corresponding ICMP Echo Request message.

Data received in an ICMP Echo Request **must** be entirely included in the resulting Echo Reply. However, if sending the Echo Reply requires intentional fragmentation that is not implemented, the datagram **must** be truncated to maximum transmission size (see Fragmentation) and sent.

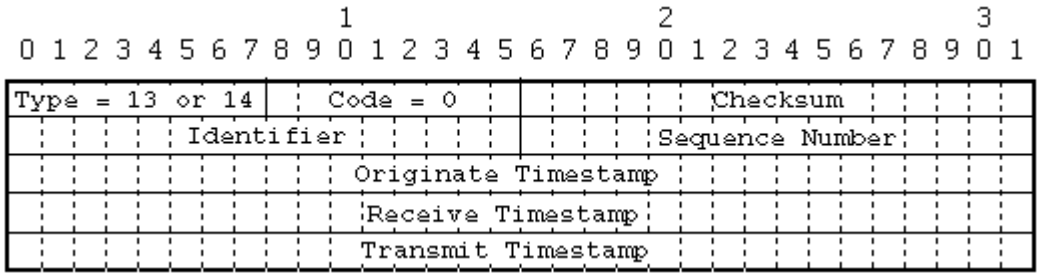
Echo Reply messages **must** be passed to the ICMP user interface, unless the corresponding Echo Request originated in the IP layer.

If a Record Route and/or Time Stamp option is received in an ICMP Echo Request, this option (these options) **should** be updated to include the current host and included in the IP header of the Echo Reply message, without "truncation". Thus, the recorded route will be for the entire round trip.

If a Source Route option is received in an ICMP Echo Request, the return route **must** be reversed and used as a Source Route option for the Echo Reply message.

RFC-792 ICMP - Message Formats

Timestamp and Timestamp Reply Message



IP Fields:

Addresses

The address of the source in a timestamp message will be the destination of the timestamp reply message. To form a timestamp reply message, the source and destination addresses are simply reversed, the type code changed to 14, and the checksum recomputed.

ICMP Fields:

Type

13 for timestamp message;
14 for timestamp reply message.

Code

0

Checksum

Identifier

If code = 0, an identifier to aid in matching timestamp and replies, may be zero.

Sequence Number

If code = 0, a sequence number to aid in matching timestamp and replies, may be zero.

Description

The data received (a timestamp) in the message is returned in the reply together with an additional timestamp. The timestamp is 32 bits of milliseconds since midnight UT. One use of these timestamps is described by Mills [5].

The Originate Timestamp is the time the sender last touched the message before sending it, the Receive Timestamp is the time the echoer first touched it on receipt, and the Transmit Timestamp is the time the echoer last touched the message on sending it.

If the time is not available in miliseconds or cannot be provided with respect to midnight UT then any time can be inserted in a timestamp provided the high order bit of the timestamp is also set to indicate this non-standard value.

The identifier and sequence number may be used by the echo sender to aid in matching the replies with the requests. For example, the identifier might be used like a port in TCP or UDP to identify a session, and the sequence number might be incremented on each request sent. The destination returns these same values in the reply.

Code 0 may be received from a gateway or a host.

See Disscussion

RFC-792 Internet Control Message Protocol

Timestamp and Timestamp Reply Discussion

A host **may** implement Timestamp and Timestamp Reply. If they are implemented, the following rules **must** be followed.

- The ICMP Timestamp server function returns a Timestamp Reply to every Timestamp message that is received. If this function is implemented, it **should** be designed for minimum variability in delay (e.g., implemented in the kernel to avoid delay in scheduling a user process).

The following cases for Timestamp are to be handled according to the corresponding rules for ICMP Echo:

- An ICMP Timestamp Request message to an IP broadcast or IP multicast address **may** be silently discarded.

- The IP source address in an ICMP Timestamp Reply **must** be the same as the specific-destination address of the corresponding Timestamp Request message.

- If a Source-route option is received in an ICMP Echo Request, the return route **must** be reversed and used as a Source Route option for the Timestamp Reply message.

- If a Record Route and/or Timestamp option is received in a Timestamp Request, this (these) option(s) **should** be updated to include the current host and included in the IP header of the Timestamp Reply message.

- Incoming Timestamp Reply messages **must** be passed up to the ICMP user interface.

The preferred form for a timestamp value (the "standard value") is in units of milliseconds since midnight Universal Time. However, it may be difficult to provide this value with millisecond resolution. For example, many systems use clocks that update only at line frequency, 50 or 60 times per second. Therefore, some latitude is allowed in a "standard value":

(a) A "standard value" **must** be updated at least 15 times per second (i.e., at most the six low-order bits of the value may be undefined).

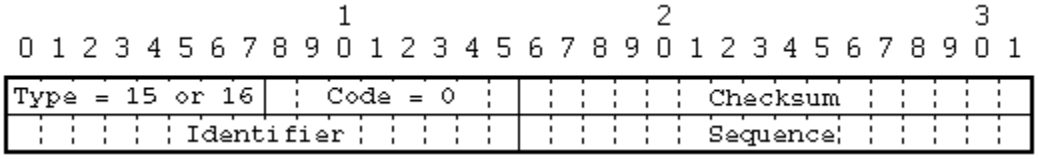
(b) The accuracy of a "standard value" **must** approximate that of operator-set CPU clocks, i.e., correct within a few minutes.

RFC-792 ICMP - Message Formats

Information Request and Information Reply Message

This message pair is obsolete and a host should not implement it. This section appears for historical reference only.

The Information Request/Reply pair was intended to support self-configuring systems such as diskless workstations, to allow them to discover their IP network numbers at boot time. However, the RARP and BOOTP protocols provide better mechanisms for a host to discover its own IP address.



IP Fields:

Addresses

The address of the source in a information request message will be the destination of the information reply message. To form a information reply message, the source and destination addresses are simply reversed, the type code changed to 16, and the checksum recomputed.

ICMP Fields:

Type

15 for information request message;
16 for information reply message.

Code

0

Checksum

Identifier

If code = 0, an identifier to aid in matching request and replies, may be zero.

Sequence Number

If code = 0, a sequence number to aid in matching request and replies, may be zero.

Description

This message may be sent with the source network in the IP header source and destination address fields zero (which means "this" network). The replying IP module should send the reply with the addresses fully specified. This message is a way for a host to find out the number of the network it is on.

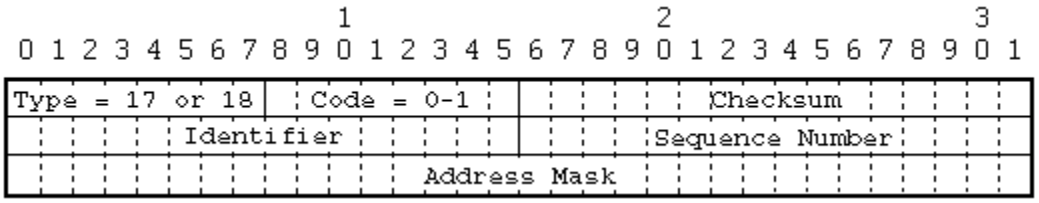
The identifier and sequence number may be used by the echo sender to aid in matching the replies with the requests. For example, the identifier might be used like a port in TCP or UDP to identify a session, and the sequence number might be incremented on each request sent. The destination returns these same values in the reply.

Code 0 may be received from a gateway or a host.

RFC-792 ICMP - Message Formats

Address Mask Request and Address Mask Reply

This message was originally defined in an appendix to "Internet Standard Subnetting Procedure" [RFC-950].



IP Fields:

Addresses

The address of the source in an address mask request message will be the destination of the address mask reply message. To form an address mask reply message, the source address of the request becomes the destination address of the reply, the source address of the reply is set to the replier's address, the type code changed to 18, the address mask value inserted into the Address Mask field, and the checksum recomputed. However, if the source address in the request message is zero, then the destination address for the reply message should denote a broadcast.

ICMP Fields:

Type

17 for address mask request message
18 for address mask reply message

Code

0 for address mask request message
1 for address mask reply message

Checksum

Identifier

An identifier to aid in matching requests and replies, may be zero.

Sequence Number

A sequence number to aid in matching requests and replies, may be zero.

Address Mask

A 32-bit mask.

Description

A gateway receiving an address mask request should return it with the address mask field set to the 32-bit mask of the bits identifying the subnet and network, for the subnet on which the request was received.

If the requesting host does not know its own IP address, it may leave the

source field zero; the reply should then be broadcast. However, this approach should be avoided if at all possible, since it increases the superfluous broadcast load on the network. Even when the replies are broadcast, since there is only one possible address mask for a subnet, there is no need to match requests with replies. The "Identifier" and "Sequence Number" fields can be ignored.

Type 17 may be received from a gateway or a host.

Type 18 may be received from a gateway, or a host acting in lieu of a gateway.

See Examples.

RFC-792 ICMP - Address Mask and Address Mask Reply

Discussion

A host **must** support the first, and **may** implement all three, of the following methods for determining the address mask(s) corresponding to its IP address(es):

(1) static configuration information;

(2) obtaining the address mask(s) dynamically as a side- effect of the system initialization process; and

(3) sending ICMP Address Mask Request(s) and receiving ICMP Address Mask Reply(s).

The choice of method to be used in a particular host **must** be configurable.

When method (3), the use of Address Mask messages, is enabled, then:

(a) When it initializes, the host **must** broadcast an Address Mask Request message on the connected network corresponding to the IP address. It **must** retransmit this message a small number of times if it does not receive an immediate Address Mask Reply.

(b) Until it has received an Address Mask Reply, the host **should** assume a mask appropriate for the address class of the IP address, i.e., assume that the connected network is not subnetted.

(c) The first Address Mask Reply message received **must** be used to set the address mask corresponding to the particular local IP address. This is true even if the first Address Mask Reply message is "unsolicited", in which case it will have been broadcast and may arrive after the host has ceased to retransmit Address Mask Requests. Once the mask has been set by an Address Mask Reply, later Address Mask Reply messages **must** be (silently) ignored.

Conversely, if Address Mask messages are disabled, then no ICMP Address Mask Requests will be sent, and any ICMP Address Mask Replies received for that local IP address **must** be (silently) ignored.

A host **should** make some reasonableness check on any address mask it installs.

A system **must not** send an Address Mask Reply unless it is an authoritative agent for address masks. An authoritative agent may be a host or a gateway, but it **must** be explicitly configured as a address mask agent. Receiving an address mask via an Address Mask Reply does not give the receiver authority and **must not** be used as the basis for issuing Address Mask Replies.

With a statically configured address mask, there **should** be an additional configuration flag that determines whether the host is to act as an authoritative agent for this mask, i.e., whether it will answer Address Mask Request messages using this mask.

If it is configured as an agent, the host **must** broadcast an Address Mask Reply for the mask on the appropriate interface when it initializes.

See System Initialization for more information about the use of Address Mask Request/Reply messages.

Hosts that casually send Address Mask Replies with invalid address masks have often been a serious nuisance. To prevent this, Address Mask Replies ought to be sent only by authoritative agents that have been selected by explicit administrative action.

When an authoritative agent receives an Address Mask Request message, it will send a

unicast Address Mask Reply to the source IP address. If the network part of this address is zero (see (a) and (b) in Addressing), the Reply will be broadcast.

Getting no reply to its Address Mask Request messages, a host will assume there is no agent and use an unsubnetted mask, but the agent may be only temporarily unreachable. An agent will broadcast an unsolicited Address Mask Reply whenever it initializes, in order to update the masks of all hosts that have initialized in the meantime.

The following reasonableness check on an address mask is suggested: the mask is not all 1 bits, and it is either zero or else the 8 highest-order bits are on.

RFC-792 ICMP - Address Mask and Address Mask Reply

Examples

These examples show how a host can find out the address mask using the ICMP Address Mask Request and Address Mask Reply messages. For the following examples, assume that address 255.255.255.255 denotes "broadcast to this physical medium" [RFC-919].

RFC-792 ICMP - Address Mask Examples

Class A Network Example

For this case, assume that the requesting host is on class A network 36.0.0.0, has address 36.40.0.123, that there is a gateway at 36.40.0.62, and that a 8-bit wide subnet field is in use, that is, the address mask is 255.255.0.0.

The most efficient method, and the one we recommend, is for a host to first discover its own address (perhaps using "RARP"), and then to send the ICMP request to 255.255.255.255:

Source address:	36.40.0.123
Destination address:	255.255.255.255
Protocol:	ICMP = 1
Type:	Address Mask Request = 17
Code:	0
Mask:	0

The gateway can then respond directly to the requesting host.

Source address:	36.40.0.62
Destination address:	36.40.0.123
Protocol:	ICMP = 1
Type:	Address Mask Reply = 18
Code:	0
Mask:	255.255.0.0

Suppose that 36.40.0.123 is a diskless workstation, and does not know even its own host number. It could send the following datagram:

Source address:	0.0.0.0
Destination address:	255.255.255.255
Protocol:	ICMP = 1
Type:	Address Mask Request = 17
Code:	0
Mask:	0

36.40.0.62 will hear the datagram, and should respond with this datagram:

Source address:	36.40.0.62
Destination address:	255.255.255.255
Protocol:	ICMP = 1
Type:	Address Mask Reply = 18
Code:	0
Mask:	255.255.0.0

Note that the gateway uses the narrowest possible broadcast to reply. Even so, the over use of broadcasts presents an unnecessary load to all hosts on the subnet, and so the use of the "anonymous" (0.0.0.0) source address must be kept to a minimum.

If broadcasting is not allowed, we assume that hosts have wired-in information about neighbor gateways; thus, 36.40.0.123 might send this datagram:

Source address:	36.40.0.123
Destination address:	36.40.0.62
Protocol:	ICMP = 1
Type:	Address Mask Request = 17
Code:	0

Mask:	0

36.40.0.62 should respond exactly as in the previous case.

Source address:	36.40.0.62
Destination address:	36.40.0.123
Protocol:	ICMP = 1
Type:	Address Mask Reply = 18
Code:	0
Mask:	255.255.0.0

RFC-792 ICMP - Adderss Mask Examples

A Class B Network Example

For this case, assume that the requesting host is on class B network 128.99.0.0, has address 128.99.4.123, that there is a gateway at 128.99.4.62, and that a 6-bit wide subnet field is in use, that is, the address mask is 255.255.252.0.

The host sends the ICMP request to 255.255.255.255:

Source address:	128.99.4.123
Destination address:	255.255.255.255
Protocol:	ICMP = 1
Type:	Address Mask Request = 17
Code:	0
Mask:	0

The gateway can then respond directly to the requesting host.

Source address:	128.99.4.62
Destination address:	128.99.4.123
Protocol:	ICMP = 1
Type:	Address Mask Reply = 18
Code:	0
Mask:	255.255.252.0

In the diskless workstation case the host sends:

Source address:	0.0.0.0
Destination address:	255.255.255.255
Protocol:	ICMP = 1
Type:	Address Mask Request = 17
Code:	0
Mask:	0

128.99.4.62 will hear the datagram, and should respond with this datagram:

Source address:	128.99.4.62
Destination address:	255.255.255.255
Protocol:	ICMP = 1
Type:	Address Mask Reply = 18
Code:	0
Mask:	255.255.252.0

If broadcasting is not allowed 128.99.4.123 sends:

Source address:	128.99.4.123
Destination address:	128.99.4.62
Protocol:	ICMP = 1
Type:	Address Mask Request = 17
Code:	0
Mask:	0

128.99.4.62 should respond exactly as in the previous case.

Source address:	128.99.4.62
Destination address:	128.99.4.123
Protocol:	ICMP = 1

Type:	Address Mask Reply = 18
Code:	0
Mask:	255.255.252.0

RFC-792 ICMP - Address Mask Examples

Class C Network Example
(illustrating non-contiguous subnet bits)

For this case, assume that the requesting host is on class C network 192.1.127.0, has address 192.1.127.19, that there is a gateway at 192.1.127.50, and that on network an 3-bit subnet field is in use (01011000), that is, the address mask is 255.255.255.88.

The host sends the ICMP request to 255.255.255.255:

Source address:	192.1.127.19
Destination address:	255.255.255.255
Protocol:	ICMP = 1
Type:	Address Mask Request = 17
Code:	0
Mask:	0

The gateway can then respond directly to the requesting host.

Source address:	192.1.127.50
Destination address:	192.1.127.19
Protocol:	ICMP = 1
Type:	Address Mask Reply = 18
Code:	0
Mask:	255.255.255.88.

In the diskless workstation case the host sends:

Source address:	0.0.0.0
Destination address:	255.255.255.255
Protocol:	ICMP = 1
Type:	Address Mask Request = 17
Code:	0
Mask:	0

192.1.127.50 will hear the datagram, and should respond with this datagram:

Source address:	192.1.127.50
Destination address:	255.255.255.255
Protocol:	ICMP = 1
Type:	Address Mask Reply = 18
Code:	0
Mask:	255.255.255.88.

If broadcasting is not allowed 192.1.127.19 sends:

Source address:	192.1.127.19
Destination address:	192.1.127.50
Protocol:	ICMP = 1
Type:	Address Mask Request = 17
Code:	0
Mask:	0

192.1.127.50 should respond exactly as in the previous case.

Source address:	192.1.127.50
Destination address:	192.1.127.19

Protocol:	ICMP = 1
Type:	Address Mask Reply = 18
Code:	0
Mask:	255.255.255.88

The 16 bit one's complement of the one's complement sum of all 16 bit words in the header. For computing the checksum, the checksum field should be zero. This checksum may be replaced in the future.

The internet header plus the first 64 bits of the original datagram's data. This data is used by the host to match the message to the appropriate process. If a higher level protocol uses port numbers, they are assumed to be in the first 64 data bits of the original datagram's data.

RFC-793 Transmission Control Protocol

DARPA Internet Program
Protocol Specification
September 1981

This section includes the original text of RFC-793 as well as many corrections, comments, and suggestions made since its publication. There are also links to other relevant documents and discussions.

RFC-793 Transmission Control Protocol - Table of Contents

Philosophy

RFC-793 Transmission Control Protocol - Table of Contents

Functional Specification

Sequence Numbers

Establishing a Connection

RFC-793 Transmission Control Protocol

Data Communication

Preface

This document describes the DoD Standard Transmission Control Protocol (TCP). There have been nine earlier editions of the ARPA TCP specification on which this standard is based, and the present text draws heavily from them. There have been many contributors to this work both in terms of concepts and in terms of text. This edition clarifies several details and removes the end-of-letter buffer-size adjustments, and redescribes the letter mechanism as a push function.

Jon Postel
Editor

RFC-793 Transmission Control Protocol

Introduction

The Transmission Control Protocol (TCP) is intended for use as a highly reliable host-to-host protocol between hosts in packet-switched computer communication networks, and in interconnected systems of such networks.

This document describes the functions to be performed by the Transmission Control Protocol, the program that implements it, and its interface to programs or users that require its services.

This version of RFC-793 incorporates a number of corrections and extensions that have been adopted by the Internet Community since its original publication in September of 1981.

RFC-793 Transmission Control Protocol - Introduction

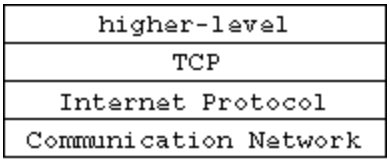
Motivation

Computer communication systems are playing an increasingly important role in military, government, and civilian environments. This document focuses its attention primarily on military computer communication requirements, especially robustness in the presence of communication unreliability and availability in the presence of congestion, but many of these problems are found in the civilian and government sector as well.

As strategic and tactical computer communication networks are developed and deployed, it is essential to provide means of interconnecting them and to provide standard interprocess communication protocols which can support a broad range of applications. In anticipation of the need for such standards, the Deputy Undersecretary of Defense for Research and Engineering has declared the Transmission Control Protocol (TCP) described herein to be a basis for DoD-wide inter-process communication protocol standardization.

TCP is a connection-oriented, end-to-end reliable protocol designed to fit into a layered hierarchy of protocols which support multi-network applications. The TCP provides for reliable inter-process communication between pairs of processes in host computers attached to distinct but interconnected computer communication networks. Very few assumptions are made as to the reliability of the communication protocols below the TCP layer. TCP assumes it can obtain a simple, potentially unreliable datagram service from the lower level protocols. In principle, the TCP should be able to operate above a wide spectrum of communication systems ranging from hard-wired connections to packet-switched or circuit-switched networks.

TCP is based on concepts first described by Cerf and Kahn in [1]. The TCP fits into a layered protocol architecture just above a basic Internet Protocol [RFC-791] which provides a way for the TCP to send and receive variable-length segments of information enclosed in internet datagram "envelopes". The internet datagram provides a means for addressing source and destination TCPs in different networks. The internet protocol also deals with any fragmentation or reassembly of the TCP segments required to achieve transport and delivery through multiple networks and interconnecting gateways. The internet protocol also carries information on the precedence, security classification and compartmentation of the TCP segments, so this information can be communicated end-to-end across multiple networks.



Protocol Layering

Much of this document is written in the context of TCP implementations which are co-resident with higher level protocols in the host computer. Some computer systems will be connected to networks via front-end computers which house the TCP and internet protocol layers, as well as network specific software. The TCP specification describes an interface to the higher level protocols which appears to be implementable even for the front-end case, as long as a suitable host-to-front end protocol is implemented.

Scope

The TCP is intended to provide a reliable process-to-process communication service in a multinetwork environment. The TCP is intended to be a host-to-host protocol in common use in multiple networks.

About this Document

This document represents a specification of the behavior required of any TCP implementation, both in its interactions with higher level protocols and in its interactions with other TCPs. The rest of this introduction offers a very brief view of the protocol interfaces and operation. The second section summarizes the philosophical basis for the TCP design. The third section offers both a detailed description of the actions required of TCP when various events occur (arrival of new segments, user calls, errors, etc.) and the details of the formats of TCP segments.

Interfaces

The TCP interfaces on one side to user or application processes and on the other side to a lower level protocol such as Internet Protocol.

The interface between an application process and the TCP is illustrated in reasonable detail. This interface consists of a set of calls much like the calls an operating system provides to an application process for manipulating files. For example, there are calls to open and close connections and to send and receive data on established connections. It is also expected that the TCP can asynchronously communicate with application programs. Although considerable freedom is permitted to TCP implementors to design interfaces which are appropriate to a particular operating system environment, a minimum functionality is required at the TCP/user interface for any valid implementation.

The interface between TCP and lower level protocol is essentially unspecified except that it is assumed there is a mechanism whereby the two levels can asynchronously pass information to each other. Typically, one expects the lower level protocol to specify this interface. TCP is designed to work in a very general environment of interconnected networks. The lower level protocol which is assumed throughout this document is the Internet Protocol [RFC-791].

Operation

As noted above, the primary purpose of the TCP is to provide reliable, securable logical circuit or connection service between pairs of processes. To provide this service on top of a less reliable internet communication system requires facilities in the following areas:

Basic Data Transfer

The TCP is able to transfer a continuous stream of octets in each direction between its users by packaging some number of octets into segments for transmission through the internet system. In general, the TCPs decide when to block and forward data at their own convenience.

Sometimes users need to be sure that all the data they have submitted to the TCP has been transmitted. For this purpose a push function is defined. To assure that data submitted to a TCP is actually transmitted the sending user indicates that it should be pushed through to the receiving user. A push causes the TCPs to promptly forward and deliver data up to that point to the receiver. The exact push point might not be visible to the receiving user and the push function does not supply a record boundary marker.

Reliability

The TCP must recover from data that is damaged, lost, duplicated, or delivered out of order by the internet communication system. This is achieved by assigning a sequence number to each octet transmitted, and requiring a positive acknowledgment (ACK) from the receiving TCP. If the ACK is not received within a timeout interval, the data is retransmitted. At the receiver, the sequence numbers are used to correctly order segments that may be received out of order and to eliminate duplicates. Damage is handled by adding a checksum to each segment transmitted, checking it at the receiver, and discarding damaged segments.

As long as the TCPs continue to function properly and the internet system does not become completely partitioned, no transmission errors will affect the correct delivery of data. TCP recovers from internet communication system errors.

Flow Control

TCP provides a means for the receiver to govern the amount of data sent by the sender. This is achieved by returning a "window" with every ACK indicating a range of acceptable sequence numbers beyond the last segment successfully received. The window indicates an allowed number of octets that the sender may transmit before receiving further permission.

Multiplexing

To allow for many processes within a single Host to use TCP communication facilities simultaneously, the TCP provides a set of addresses or ports within each host. Concatenated with the network and host addresses from the internet communication layer, this forms a socket. A pair of sockets uniquely identifies each connection. That is, a socket may be simultaneously used in multiple connections.

The binding of ports to processes is handled independently by each Host. However, it proves useful to attach frequently used processes (e.g., a "logger" or timesharing service) to fixed sockets which are made known to the public. These services can then be accessed through the known addresses. Establishing and learning the port addresses of other processes may involve more dynamic mechanisms.

Connections

The reliability and flow control mechanisms described above require that TCPs initialize and maintain certain status information for each data stream. The combination of this information, including sockets, sequence numbers, and window sizes, is called a connection. Each connection is uniquely specified by a pair of sockets identifying its two sides.

When two processes wish to communicate, their TCP's must first establish a connection (initialize the status information on each side). When their communication is complete, the connection is terminated or closed to free the resources for other uses.

Since connections must be established between unreliable hosts and over the unreliable internet communication system, a handshake mechanism with clock-based sequence numbers is used to avoid erroneous initialization of connections.

Precedence and Security

The users of TCP may indicate the security and precedence of their communication. Provision is made for default values to be used when these features are not needed.

RFC-793 Transmission Control Protocol - Philosophy

Elements of the Internetwork System

The internetwork environment consists of hosts connected to networks which are in turn interconnected via gateways. It is assumed here that the networks may be either local networks (e.g., the ETHERNET) or large networks (e.g., the ARPANET), but in any case are based on packet switching technology. The active agents that produce and consume messages are processes. Various levels of protocols in the networks, the gateways, and the hosts support an interprocess communication system that provides two-way data flow on logical connections between process ports.

The term packet is used generically here to mean the data of one transaction between a host and its network. The format of data blocks exchanged within the a network will generally not be of concern to us.

Hosts are computers attached to a network, and from the communication network's point of view, are the sources and destinations of packets. Processes are viewed as the active elements in host computers (in accordance with the fairly common definition of a process as a program in execution). Even terminals and files or other I/O devices are viewed as communicating with each other through the use of processes. Thus, all communication is viewed as inter-process communication.

Since a process may need to distinguish among several communication streams between itself and another process (or processes), we imagine that each process may have a number of ports through which it communicates with the ports of other processes.

Model of Operation

Processes transmit data by calling on the TCP and passing buffers of data as arguments. The TCP packages the data from these buffers into segments and calls on the internet module to transmit each segment to the destination TCP. The receiving TCP places the data from a segment into the receiving user's buffer and notifies the receiving user. The TCPs include control information in the segments which they use to ensure reliable ordered data transmission.

The model of internet communication is that there is an internet protocol module associated with each TCP which provides an interface to the local network. This internet module packages TCP segments inside internet datagrams and routes these datagrams to a destination internet module or intermediate gateway. To transmit the datagram through the local network, it is embedded in a local network packet.

The packet switches may perform further packaging, fragmentation, or other operations to achieve the delivery of the local packet to the destination internet module.

At a gateway between networks, the internet datagram is "unwrapped" from its local packet and examined to determine through which network the internet datagram should travel next. The internet datagram is then "wrapped" in a local packet suitable to the next network and routed to the next gateway, or to the final destination.

A gateway is permitted to break up an internet datagram into smaller internet datagram fragments if this is necessary for transmission through the next network. To do this, the gateway produces a set of internet datagrams; each carrying a fragment. Fragments may be further broken into smaller fragments at subsequent gateways. The internet datagram fragment format is designed so that the destination internet module can reassemble fragments into internet datagrams.

A destination internet module unwraps the segment from the datagram (after reassembling the datagram, if necessary) and passes it to the destination TCP.

This simple model of the operation glosses over many details. One important feature is the type of service. This provides information to the gateway (or internet module) to guide it in selecting the service parameters to be used in traversing the next network. Included in the type of service information is the precedence of the datagram. Datagrams may also carry security information to permit host and gateways that operate in multilevel secure environments to properly segregate datagrams for security considerations.

The Host Environment

The TCP is assumed to be a module in an operating system. The users access the TCP much like they would access the file system. The TCP may call on other operating system functions, for example, to manage data structures. The actual interface to the network is assumed to be controlled by a device driver module. The TCP does not call on the network device driver directly, but rather calls on the internet datagram protocol module which may in turn call on the device driver.

The mechanisms of TCP do not preclude implementation of the TCP in a front-end processor. However, in such an implementation, a host-to-front-end protocol must provide the functionality to support the type of TCP-user interface described in this document.

Interfaces

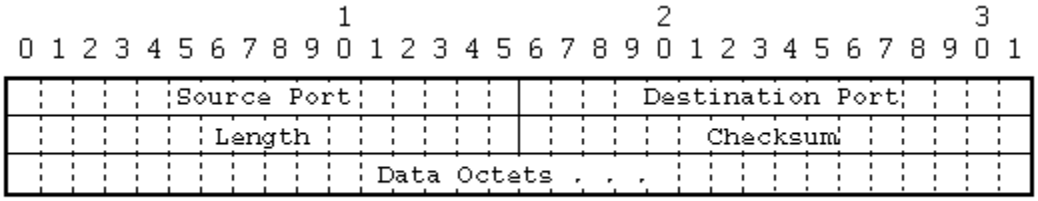
The TCP/user interface provides for calls made by the user on the TCP to OPEN or CLOSE a connection, to SEND or RECEIVE data, or to obtain STATUS about a connection. These calls are like other calls from user programs on the operating system, for example, the calls to open, read from, and close a file.

The TCP/internet interface provides calls to send and receive datagrams addressed to TCP modules in hosts anywhere in the internet system. These calls have parameters for passing the address, type of service, precedence, security, and other control information.

RFC-793 Transmission Control Protocol - Philosophy

Relation to Other Protocols

The following diagram illustrates the place of the TCP in the protocol hierarchy:



User Datagram Header Format

It is expected that the TCP will be able to support higher level protocols efficiently. It should be easy to interface higher level protocols like the ARPANET Telnet or AUTODIN II THP to the TCP.

RFC-793 Transmission Control Protocol - Philosophy

Reliable Communication

A stream of data sent on a TCP connection is delivered reliably and in order at the destination.

Transmission is made reliable via the use of sequence numbers and acknowledgments. Conceptually, each octet of data is assigned a sequence number. The sequence number of the first octet of data in a segment is transmitted with that segment and is called the segment sequence number. Segments also carry an acknowledgment number which is the sequence number of the next expected data octet of transmissions in the reverse direction. When the TCP transmits a segment containing data, it puts a copy on a retransmission queue and starts a timer; when the acknowledgment for that data is received, the segment is deleted from the queue. If the acknowledgment is not received before the timer runs out, the segment is retransmitted.

An acknowledgment by TCP does not guarantee that the data has been delivered to the end user, but only that the receiving TCP has taken the responsibility to do so.

To govern the flow of data between TCPs, a flow control mechanism is employed. The receiving TCP reports a "window" to the sending TCP. This window specifies the number of octets, starting with the acknowledgment number, that the receiving TCP is currently prepared to receive.

RFC-793 Transmission Control Protocol - Philosophy

Connection Establishment and Clearing

To identify the separate data streams that a TCP may handle, the TCP provides a port identifier. Since port identifiers are selected independently by each TCP they might not be unique. To provide for unique addresses within each TCP, we concatenate an internet address identifying the TCP with a port identifier to create a socket which will be unique throughout all networks connected together.

A connection is fully specified by the pair of sockets at the ends. A local socket may participate in many connections to different foreign sockets. A connection can be used to carry data in both directions, that is, it is "full duplex".

TCPs are free to associate ports with processes however they choose. However, several basic concepts are necessary in any implementation. There must be well-known sockets which the TCP associates only with the "appropriate" processes by some means. We envision that processes may "own" ports, and that processes can initiate connections only on the ports they own. (Means for implementing ownership is a local issue, but we envision a Request Port user command, or a method of uniquely allocating a group of ports to a given process, e.g., by associating the high order bits of a port name with a given process.)

A connection is specified in the OPEN call by the local port and foreign socket arguments. In return, the TCP supplies a (short) local connection name by which the user refers to the connection in subsequent calls. There are several things that must be remembered about a connection. To store this information we imagine that there is a data structure called a Transmission Control Block (TCB). One implementation strategy would have the local connection name be a pointer to the TCB for this connection. The OPEN call also specifies whether the connection establishment is to be actively pursued, or to be passively waited for.

A passive OPEN request means that the process wants to accept incoming connection requests rather than attempting to initiate a connection. Often the process requesting a passive OPEN will accept a connection request from any caller. In this case a foreign socket of all zeros is used to denote an unspecified socket. Unspecified foreign sockets are allowed only on passive OPENs.

A service process that wished to provide services for unknown other processes would issue a passive OPEN request with an unspecified foreign socket. Then a connection could be made with any process that requested a connection to this local socket. It would help if this local socket were known to be associated with this service.

Well-known sockets are a convenient mechanism for a priori associating a socket address with a standard service. For instance, the "Telnet-Server" process is permanently assigned to a particular socket, and other sockets are reserved for File Transfer, Remote Job Entry, Text Generator, Echoer, and Sink processes (the last three being for test purposes). A socket address might be reserved for access to a "Look-Up" service which would return the specific socket at which a newly created service would be provided. The concept of a well-known socket is part of the TCP specification, but the assignment of sockets to services is outside this specification. (See Assigned Numbers.)

Processes can issue passive OPENs and wait for matching active OPENs from other processes and be informed by the TCP when connections have been established. Two processes which issue active OPENs to each other at the same time will be correctly connected. This flexibility is critical for the support of distributed computing in which components act asynchronously with respect to each other.

There are two principal cases for matching the sockets in the local passive OPENs and an

foreign active OPENs. In the first case, the local passive OPENs has fully specified the foreign socket. In this case, the match must be exact. In the second case, the local passive OPENs has left the foreign socket unspecified. In this case, any foreign socket is acceptable as long as the local sockets match. Other possibilities include partially restricted matches.

If there are several pending passive OPENs (recorded in TCBs) with the same local socket, an foreign active OPEN will be matched to a TCB with the specific foreign socket in the foreign active OPEN, if such a TCB exists, before selecting a TCB with an unspecified foreign socket.

The procedures to establish connections utilize the synchronize (SYN) control flag and involves an exchange of three messages. This exchange has been termed a three-way hand shake [3].

A connection is initiated by the rendezvous of an arriving segment containing a SYN and a waiting TCB entry each created by a user OPEN command. The matching of local and foreign sockets determines when a connection has been initiated. The connection becomes "established" when sequence numbers have been synchronized in both directions.

The clearing of a connection also involves the exchange of segments, in this case carrying the FIN control flag.

Well-Known Ports

TCP reserves port numbers in the range 0-255 for "well-known" ports, used to access services that are standardized across the Internet. The remainder of the port space can be freely allocated to application processes. Current well-known port definitions are listed in the RFC entitled "Assigned Numbers". A prerequisite for defining a new well- known port is an RFC documenting the proposed service in enough detail to allow new implementations.

Some systems extend this notion by adding a third subdivision of the TCP port space: reserved ports, which are generally used for operating-system-specific services. For example, reserved ports might fall between 256 and some system-dependent upper limit. Some systems further choose to protect well-known and reserved ports by permitting only privileged users to open TCP connections with those port values. This is perfectly reasonable as long as the host does not assume that all hosts protect their low-numbered ports in this manner.

RFC-793 Transmission Control Protocol - Philosophy

Data Communication

The data that flows on a connection may be thought of as a stream of octets. The sending user indicates in each SEND call whether the data in that call (and any preceeding calls) should be immediately pushed through to the receiving user by the setting of the PUSH flag, if this facility is implemented. If there is no facility to set the PUSH flag in the SEND call, then the sending TCP (1) **Must** not buffer data indefinitely, and (2) **must** set the PUSH flag in the last buffered segment (i.e., when there is no more data to be sent).

A sending TCP is allowed to collect data from the sending user and to send that data in segments at its own convenience, until the push function is signaled, then it must send all unsent data. When a receiving TCP sees the PUSH flag, it must not wait for more data from the sending TCP before passing the data to the receiving process.

There is no necessary relationship between push functions and segment boundaries. The data in any particular segment may be the result of a single SEND call, in whole or part, or of multiple SEND calls.

The purpose of push function and the PUSH flag is to push data through from the sending user to the receiving user. It does not provide a record service. The sending TCP **should** collapse successive PSH bits when it packetizes data, to send the largest possible segment.

There is a coupling between the push function and the use of buffers of data that cross the TCP/user interface. Each time a PUSH flag is associated with data placed into the receiving user's buffer, the buffer is returned to the user for processing even if the buffer is not filled. If data arrives that fills the user's buffer before a PUSH is seen, the data is passed to the user in buffer size units.

TCP also provides a means to communicate to the receiver of data that at some point further along in the data stream than the receiver is currently reading there is urgent data. TCP does not attempt to define what the user specifically does upon being notified of pending urgent data, but the general notion is that the receiving process will take action to process the urgent data quickly.

Pushing Data Through

An application program is logically required to set the PUSH flag in a SEND call whenever it needs to force delivery of the data to avoid a communication deadlock. However, a TCP **should** send a maximum-sized segment whenever possible, to improve performance.

Discussion

When the PUSH flag is not implemented on SEND calls, i.e., when the application/TCP interface uses a pure streaming model, responsibility for aggregating any tiny data fragments to form reasonable sized segments is partially borne by the application layer.

Generally, an interactive application protocol must set the PUSH flag at least in the last SEND call in each command or response sequence. A bulk transfer protocol like FTP should set the PUSH flag on the last segment of a file or when necessary to prevent buffer deadlock.

At the receiver, the PSH bit forces buffered data to be delivered to the application (even if less than a full buffer has been received). Conversely, the lack of a PSH bit can be used to avoid unnecessary wakeup calls to the application process; this can be an important performance optimization for large timesharing hosts. Passing the PSH bit to the receiving application allows an analogous optimization within the application.

Precedence and Security

The TCP makes use of the internet protocol type of service field and security option to provide precedence and security on a per connection basis to TCP users. Not all TCP modules will necessarily function in a multilevel secure environment; some may be limited to unclassified use only, and others may operate at only one security level and compartment. Consequently, some TCP implementations and services to users may be limited to a subset of the multilevel secure case.

TCP modules which operate in a multilevel secure environment must properly mark outgoing segments with the security, compartment, and precedence. Such TCP modules must also provide to their users or higher level protocols such as Telnet or THP an interface to allow them to specify the desired security level, compartment, and precedence of connections.

RFC-793 Transmission Control Protocol - Philosophy

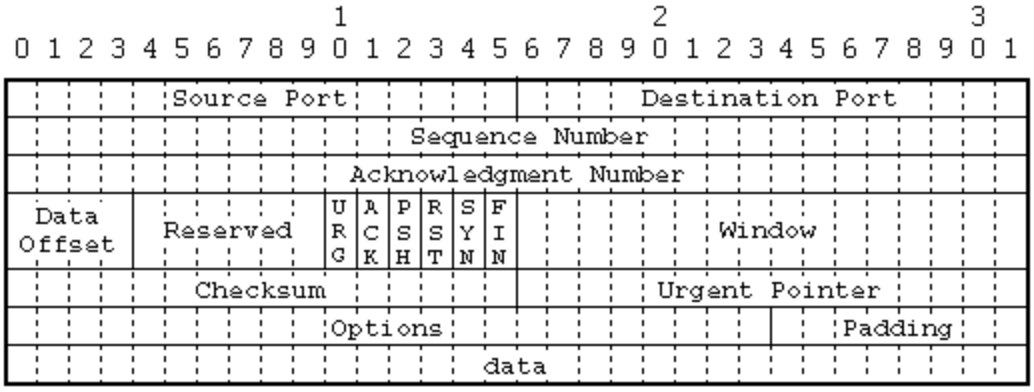
Robustness Principle

TCP implementations will follow a general principle of robustness: be conservative in what you do, be liberal in what you accept from others.

RFC-793 Transmission Control Protocol - Functional Specification

Header Format

TCP segments are sent as internet datagrams. The Internet Protocol header carries several information fields, including the source and destination host addresses. A TCP header follows the internet header, supplying information specific to the TCP protocol. This division allows for the existence of host level protocols other than TCP.



TCP Header Format

Use the Browse button to see a description of each header field.

RFC-793 Transmission Control Protocol - Header

Source Port: 16 bits

The source port number.

Destination Port: 16 bits

The destination port number.

RFC-793 Transmission Control Protocol - Header

Sequence Number: 32 bits

The sequence number of the first data octet in this segment (except when SYN is present). If SYN is present the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1.

Acknowledgment Number: 32 bits

If the ACK control bit is set this field contains the value of the next sequence number the sender of the segment is expecting to receive. Once a connection is established this is always sent.

RFC-793 Transmission Control Protocol - Header

Data Offset: 4 bits

The number of 32 bit words in the TCP Header. This indicates where the data begins. The TCP header (even one including options) is an integral number of 32 bits long.

Reserved: 6 bits

Reserved for future use. Must be zero.

RFC-793 Transmission Control Protocol - Header

Control Bits: 6 bits (from left to right):

URG: Urgent Pointer field significant
ACK: Acknowledgment field significant
PSH: Push Function
RST: Reset the connection
SYN: Synchronize sequence numbers
FIN: No more data from sender

Window: 16 bits

The number of data octets beginning with the one indicated in the acknowledgment field which the sender of this segment is willing to accept.

The window size **must** be treated as an unsigned number, or else large window sizes will appear like negative windows and TCP will not work. It is RECOMMENDED that implementations reserve 32-bit fields for the send and receive window sizes in the connection record and do all window computations with 32 bits.

Discussion

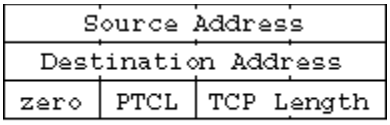
It is known that the window field in the TCP header is too small for high-speed, long-delay paths. Experimental TCP options have been defined to extend the window size; see for example RFC-1072 "TCP Extensions for Long Delay Paths". In anticipation of the adoption of such an extension, TCP implementors should treat windows as 32 bits.

RFC-793 Transmission Control Protocol - Header

Checksum: 16 bits

The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header and text. If a segment contains an odd number of header and text octets to be checksummed, the last octet is padded on the right with zeros to form a 16 bit word for checksum purposes. The pad is not transmitted as part of the segment. While computing the checksum, the checksum field itself is replaced with zeros.

The checksum also covers a 96 bit pseudo header conceptually prefixed to the TCP header. This pseudo header contains the Source Address, the Destination Address, the Protocol, and TCP length. This gives the TCP protection against misrouted segments. This information is carried in the Internet Protocol and is transferred across the TCP/Network interface in the arguments or results of calls by the TCP on the IP.



Pseudo Header

The TCP Length is the TCP header length plus the data length in octets (this is not an explicitly transmitted quantity, but is computed), and it does not count the 12 octets of the pseudo header.

Unlike the UDP checksum, the TCP checksum is never optional. The sender **must** generate it and the receiver **must** check it.

RFC-793 Transmission Control Protocol - Header

Urgent Pointer: 16 bits

This field communicates the current value of the urgent pointer as a positive offset from the sequence number in this segment. The urgent pointer points to the sequence number of the last octet of urgent data. This field is only be interpreted in segments with the URG control bit set.

A TCP **must** support a sequence of urgent data of any length.

A TCP **must** inform the application layer asynchronously whenever it receives an Urgent pointer and there was previously no pending urgent data, or whenever the Urgent pointer advances in the data stream. There **must** be a way for the application to learn how much urgent data remains to be read from the connection, or at least to determine whether or not more urgent data remains to be read.

Discussion
Although the Urgent mechanism may be used for any application, it is normally used to send "interrupt"- type commands to a Telnet program (see Telnet Synch Sequence.

The asynchronous or "out-of-band" notification will allow the application to go into "urgent mode", reading data from the TCP connection. This allows control commands to be sent to an application whose normal input buffers are full of unprocessed data.

Implementation
The generic ERROR-REPORT() upcall described in Asynchronous Reports is a possible mechanism for informing the application of the arrival of urgent data.

RFC-793 Transmission Control Protocol - Header

Options: variable

Options may occupy space at the end of the TCP header and are a multiple of 8 bits in length. All options are included in the checksum. An option may begin on any octet boundary. There are two cases for the format of an option:

Case 1: A single octet of option-kind.
Case 2: An octet of option-kind, an octet of option-length, and the actual option-data octets.

The option-length counts the two octets of option-kind and option-length as well as the option-data octets.

Note that the list of options may be shorter than the data offset field might imply. The content of the header beyond the End-of-Option option must be header padding (i.e., zero).

Currently defined options include (kind indicated in octal):

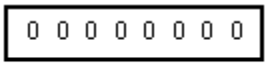
Kind	Length	Meaning
0	-	End of Option List
1	-	No Operation
2	4	Maximum Segment Size

A TCP **must** be able to receive a TCP option in any segment. A TCP **must** ignore without error any TCP option it does not implement, assuming that the option has a length field (all TCP options defined in the future will have length fields). TCP **must** be prepared to handle an illegal option length (e.g., zero) without crashing; a suggested procedure is to reset the connection and log the reason.

Padding: variable

The TCP header padding is used to ensure that the TCP header ends and data begins on a 32 bit boundary. The padding is composed of zeros.

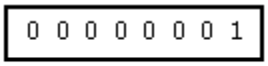
End of Option List



Kind = 0

This option code indicates the end of the option list. This might not coincide with the end of the TCP header according to the Data Offset field. This is used at the end of all options, not the end of each option, and need only be used if the end of the options would not otherwise coincide with the end of the TCP header.

No-Operation

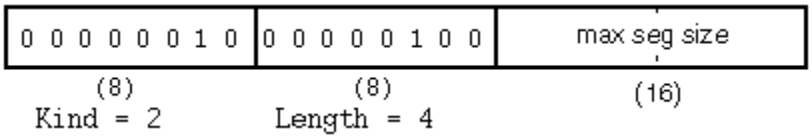


Kind = 1

This option code may be used between options, for example, to align the beginning of a subsequent option on a word boundary. There is no guarantee that senders will use this option, so receivers must be prepared to process options even if they do not begin on a word boundary.

RFC-793 Transmission Control Protocol - Functional Specification

Maximum Segment Size



Maximum Segment Size Option Data: 16 bits

If this option is present, then it communicates the maximum receive segment size at the TCP which sends this segment. This field must only be sent in the initial connection request (i.e., in segments with the SYN control bit set). If this option is not used, any segment size is allowed.

TCP **must** implement both sending and receiving the Maximum Segment Size option .

TCP **should** send an MSS (Maximum Segment Size) option in every SYN segment when its receive MSS differs from the default 536, and **may** send it always.

If an MSS option is not received at connection setup, TCP **must** assume a default send MSS of 536 (576-40).

The maximum size of a segment that TCP really sends, the "effective send MSS," **must** be the smaller of the send MSS (which reflects the available reassembly buffer size at the remote host) and the largest size permitted by the IP layer:

Eff.snd.MSS = min(SendMSS+20, MMS_S) - TCPhdrsize - IPoptionsize

where:

* SendMSS is the MSS value received from the remote host, or the default 536 if no MSS option is received.
* MMS_S is the maximum size for a transport-layer message that TCP may send.
* TCPhdrsize is the size of the TCP header; this is normally 20, but may be larger if TCP options are to be sent.
* IPoptionsize is the size of any IP options that TCP will pass to the IP layer with the current message.

The MSS value to be sent in an MSS option must be less than or equal to:

MMS_R - 20

where MMS_R is the maximum size for a transport-layer message that can be received (and reassembled). TCP obtains MMS_R and MMS_S from the IP layer; see the generic call GET_MAXSIZES.

Discussion

The choice of TCP segment size has a strong effect on performance. Larger segments increase throughput by amortizing header size and per-datagram processing overhead over more data bytes; however, if the packet is so large that it causes IP fragmentation, efficiency drops sharply if any fragments are lost [IP:9].

Some TCP implementations send an MSS option only if the destination host is on a non-connected network. However, in general the TCP layer may not have the appropriate information to make this decision, so it is preferable to leave

to the IP layer the task of determining a suitable MTU for the Internet path. We therefore recommend that TCP always send the option (if not 536) and that the IP layer determine MMS_R and Internet/Transport Layer Interface. A proposed IP-layer mechanism to measure the MTU would then modify the IP layer without changing TCP.

RFC-793 Transmission Control Protocol - Functional Specification

Terminology

Before we can discuss very much about the operation of the TCP we need to introduce some detailed terminology. The maintenance of a TCP connection requires the remembering of several variables. We conceive of these variables being stored in a connection record called a Transmission Control Block or TCB. Among the variables stored in the TCB are the local and remote socket numbers, the security and precedence of the connection, pointers to the user's send and receive buffers, pointers to the retransmit queue and to the current segment. In addition several variables relating to the send and receive sequence numbers are stored in the TCB.

Send Sequence Variables

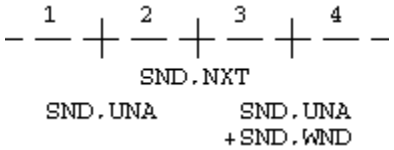
SND.UNA	send unacknowledged
SND.NXT	send next
SND.WND	send window
SND.UP	send urgent pointer
SND.WL1	segment sequence number used for last window update
SND.WL2	segment acknowledgment number used for last window update
ISS	initial send sequence number

Receive Sequence Variables

RCV.NXT	receive next
RCV.WND	receive window
RCV.UP	receive urgent pointer
IRS	initial receive sequence number

The following diagrams may help to relate some of these variables to the sequence space.

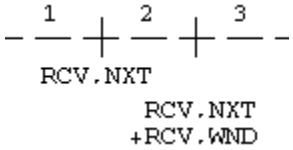
Send Sequence Space



1 old sequence numbers which have been acknowledged
2 sequence numbers of unacknowledged data
3 sequence numbers allowed for new data transmission
4 future sequence numbers which are not yet allowed

The send window is the portion of the sequence space labeled 3 in the above figure.

Receive Sequence Space



1 old sequence numbers which have been acknowledged
2 sequence numbers allowed for new reception
3 future sequence numbers which are not yet allowed

The receive window is the portion of the sequence space labeled 2 in figure 5.

There are also some variables used frequently in the discussion that take their values from the fields of the current segment.

Current Segment Variables

SEG.SEQ	segment sequence number
SEG.ACK	segment acknowledgment number
SEG.LEN	segment length
SEG.WND	segment window
SEG.UP	segment urgent pointer
SEG.PRC	segment precedence value

A connection progresses through a series of states during its lifetime. The states are: LISTEN, SYN-SENT, SYN-RECEIVED, ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT, and the fictional state CLOSED. CLOSED is fictional because it represents the state when there is no TCB, and therefore, no connection. Briefly the meanings of the states are:

A TCP connection progresses from one state to another in response to events. The events are the user calls, OPEN, SEND, RECEIVE, CLOSE, ABORT, and STATUS; the incoming segments, particularly those containing the SYN, ACK, RST and FIN flags; and timeouts.

LISTEN	represents waiting for a connection request from any remote TCP and port.

SYN-SENT represents waiting for a matching connection request after having sent a connection request.

SYN-RECEIVED represents waiting for a confirming connection request acknowledgment after having both received and sent a connection request.

ESTABLISHED represents an open connection, data received can be delivered to the user. The normal state for the data transfer phase of the connection.

FIN-WAIT-1 represents waiting for a connection termination request from the remote TCP, or an acknowledgment of the connection termination request previously sent.

FIN-WAIT-2 represents waiting for a connection termination request from the remote TCP.

CLOSE-WAIT represents waiting for a connection termination request from the local user.

CLOSING represents waiting for a connection termination request acknowledgment from the remote TCP.

LAST-ACK represents waiting for an acknowledgment of the connection termination request previously sent to the remote TCP (which includes an acknowledgment of its connection termination request).

TIME-WAIT represents waiting for enough time to pass to be sure the remote TCP received the acknowledgment of its connection termination request.

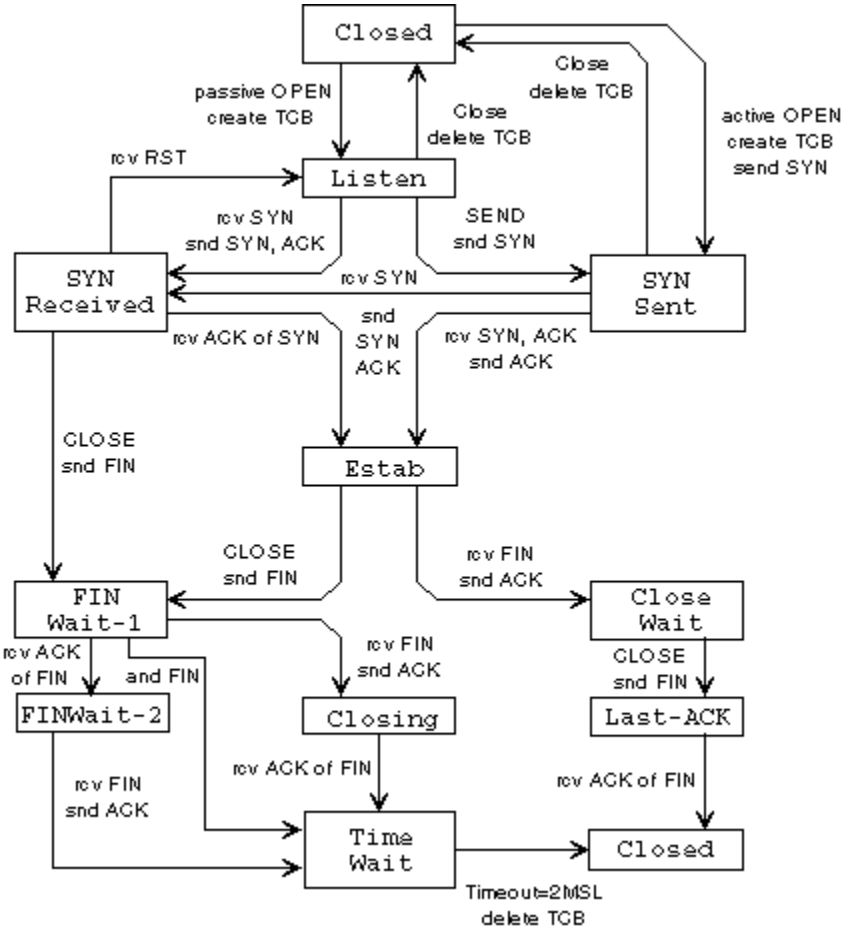
CLOSED represents no connection state at all.

RFC-793 Transmission Control Protocol - Functional Specification

State Diagram

The state diagram below illustrates only state changes, together with the causing events and resulting actions, but addresses neither error conditions nor actions which are not connected with state changes. In a later section, more detail is offered with respect to the reaction of the TCP to events.

NOTE BENE: this diagram is only a summary and must not be taken as the total specification.



Sequence Numbers

A fundamental notion in the design is that every octet of data sent over a TCP connection has a sequence number. Since every octet is sequenced, each of them can be acknowledged. The acknowledgment mechanism employed is cumulative so that an acknowledgment of sequence number X indicates that all octets up to but not including X have been received. This mechanism allows for straight-forward duplicate detection in the presence of retransmission. Numbering of octets within a segment is that the first data octet immediately following the header is the lowest numbered, and the following octets are numbered consecutively.

It is essential to remember that the actual sequence number space is finite, though very large. This space ranges from 0 to 2**32 - 1. Since the space is finite, all arithmetic dealing with sequence numbers must be performed modulo 2**32. This unsigned arithmetic preserves the relationship of sequence numbers as they cycle from 2**32 - 1 to 0 again. There are some subtleties to computer modulo arithmetic, so great care should be taken in programming the comparison of such values. The symbol "=<" means "less than or equal" (modulo 2**32).

The typical kinds of sequence number comparisons which the TCP must perform include:

(a) Determining that an acknowledgment refers to some sequence number sent but not yet acknowledged.

(b) Determining that all sequence numbers occupied by a segment have been acknowledged (e.g., to remove the segment from a retransmission queue).

(c) Determining that an incoming segment contains sequence numbers which are expected (i.e., that the segment "overlaps" the receive window).

RFC-793 Transmission Control Protocol - Sequence Numbers

Processing Acknowledgements

In response to sending data the TCP will receive acknowledgments. The following comparisons are needed to process the acknowledgments.

SND.UNA =	oldest unacknowledged sequence number
SND.NXT =	next sequence number to be sent
SEG.ACK =	acknowledgment from the receiving TCP (next sequence number expected by the receiving TCP)
SEG.SEQ =	first sequence number of a segment
SEG.LEN =	the number of octets occupied by the data in the segment (counting SYN and FIN)
SEG.SEQ+ SEG.LEN-1 =	last sequence number of a segment

A new acknowledgment (called an "acceptable ack"), is one for which the inequality below holds:

SND.UNA < SEG.ACK =< SND.NXT

A segment on the retransmission queue is fully acknowledged if the sum of its sequence number and length is less or equal than the acknowledgment value in the incoming segment.

When data is received the following comparisons are needed:

RCV.NXT =	next sequence number expected on an incoming segments, and is the left or lower edge of the receive window
RCV.NXT+ RCV.WND-1 =	last sequence number expected on an incoming segment, and is the right or upper edge of the receive window
SEG.SEQ =	first sequence number occupied by the incoming segment
SEG.SEQ+ SEG.LEN-1 =	last sequence number occupied by the incoming segment

RFC-793 Transmission Control Protocol - Sequence Numbers

Validity of Received Segments

A segment is judged to occupy a portion of valid receive sequence space if

RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND

or

RCV.NXT =< SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.WND

The first part of this test checks to see if the beginning of the segment falls in the window, the second part of the test checks to see if the end of the segment falls in the window; if the segment passes either part of the test it contains data in the window.

Actually, it is a little more complicated than this. Due to zero windows and zero length segments, we have four cases for the acceptability of an incoming segment:

Segment Length	Receive Window	Test
0	0	SEG.SEQ = RCV.NXT
0	>0	RCV.NXT <= SEG.SEQ < RCV.NXT + RCV.WND
>0	0	not acceptable
>0	>0	RCV.NXT <= SEG.SEQ < RCV.NXT + RCV.WND or RCV.NXT <= SEG.SEQ + SEG.LEN - 1 < RCV.NXT+RCV.WND

Note that when the receive window is zero no segments should be acceptable except ACK segments. Thus, it is be possible for a TCP to maintain a zero receive window while transmitting data and receiving ACKs. However, even when the receive window is zero, a TCP must process the RST and URG fields of all incoming segments.

We have taken advantage of the numbering scheme to protect certain control information as well. This is achieved by implicitly including some control flags in the sequence space so they can be retransmitted and acknowledged without confusion (i.e., one and only one copy of the control will be acted upon). Control information is not physically carried in the segment data space. Consequently, we must adopt rules for implicitly assigning sequence numbers to control. The SYN and FIN are the only controls requiring this protection, and these controls are used only at connection opening and closing. For sequence number purposes, the SYN is considered to occur before the first actual data octet of the segment in which it occurs, while the FIN is considered to occur after the last actual data octet in a segment in which it occurs. The segment length (SEG.LEN) includes both data and sequence space occupying controls. When a SYN is present then SEG.SEQ is the sequence number of the SYN.

RFC-793 Transmission Control Protocol - Sequence Numbers

Initial Sequence Number Selection

The protocol places no restriction on a particular connection being used over and over again. A connection is defined by a pair of sockets. New instances of a connection will be referred to as incarnations of the connection. The problem that arises from this is -- "how does the TCP identify duplicate segments from previous incarnations of the connection?" This problem becomes apparent if the connection is being opened and closed in quick succession, or if the connection breaks with loss of memory and is then reestablished.

To avoid confusion we must prevent segments from one incarnation of a connection from being used while the same sequence numbers may still be present in the network from an earlier incarnation. We want to assure this, even if a TCP crashes and loses all knowledge of the sequence numbers it has been using. When new connections are created, an initial sequence number (ISN) generator is employed which selects a new 32 bit ISN. The generator is bound to a (possibly fictitious) 32 bit clock whose low order bit is incremented roughly every 4 microseconds. Thus, the ISN cycles approximately every 4.55 hours. Since we assume that segments will stay in the network no more than the Maximum Segment Lifetime (MSL) and that the MSL is less than 4.55 hours we can reasonably assume that ISN's will be unique.

For each connection there is a send sequence number and a receive sequence number. The initial send sequence number (ISS) is chosen by the data sending TCP, and the initial receive sequence number (IRS) is learned during the connection establishing procedure.

For a connection to be established or initialized, the two TCPs must synchronize on each other's initial sequence numbers. This is done in an exchange of connection establishing segments carrying a control bit called "SYN" (for synchronize) and the initial sequence numbers. As a shorthand, segments carrying the SYN bit are also called "SYNs". Hence, the solution requires a suitable mechanism for picking an initial sequence number and a slightly involved handshake to exchange the ISN's.

The synchronization requires each side to send it's own initial sequence number and to receive a confirmation of it in acknowledgment from the other side. Each side must also receive the other side's initial sequence number and send a confirming acknowledgment.

1) A --> B SYN my sequence number is X
2) A <-- B ACK your sequence number is X
3) A <-- B SYN my sequence number is Y
4) A --> B ACK your sequence number is Y

Because steps 2 and 3 can be combined in a single message this is called the three way (or three message) handshake.

A three way handshake is necessary because sequence numbers are not tied to a global clock in the network, and TCPs may have different mechanisms for picking the ISN's. The receiver of the first SYN has no way of knowing whether the segment was an old delayed one or not, unless it remembers the last sequence number used on the connection (which is not always possible), and so it must ask the sender to verify this SYN. The three way handshake and the advantages of a clock-driven scheme are discussed in [3].

Knowing When to Keep Quiet

To be sure that a TCP does not create a segment that carries a sequence number which may be duplicated by an old segment remaining in the network, the TCP must keep quiet for a maximum segment lifetime (MSL) before assigning any sequence numbers upon starting up or recovering from a crash in which memory of sequence numbers in use was lost. For this specification the MSL is taken to be 2 minutes. This is an engineering choice, and may be changed if experience indicates it is desirable to do so. Note that if a TCP is reinitialized in some sense, yet retains its memory of sequence numbers in use, then it need not wait at all; it must only be sure to use sequence numbers larger than those recently used.

RFC-793 Transmission Control Protocol - Sequence Numbers

The TCP Quiet Time Concept

This specification provides that hosts which "crash" without retaining any knowledge of the last sequence numbers transmitted on each active (i.e., not closed) connection shall delay emitting any TCP segments for at least the agreed Maximum Segment Lifetime (MSL) in the internet system of which the host is a part. In the paragraphs below, an explanation for this specification is given. TCP implementors may violate the "quiet time" restriction, but only at the risk of causing some old data to be accepted as new or new data rejected as old duplicated by some receivers in the internet system.

TCPs consume sequence number space each time a segment is formed and entered into the network output queue at a source host. The duplicate detection and sequencing algorithm in the TCP protocol relies on the unique binding of segment data to sequence space to the extent that sequence numbers will not cycle through all 2**32 values before the segment data bound to those sequence numbers has been delivered and acknowledged by the receiver and all duplicate copies of the segments have "drained" from the internet. Without such an assumption, two distinct TCP segments could conceivably be assigned the same or overlapping sequence numbers, causing confusion at the receiver as to which data is new and which is old. Remember that each segment is bound to as many consecutive sequence numbers as there are octets of data in the segment.

Under normal conditions, TCPs keep track of the next sequence number to emit and the oldest awaiting acknowledgment so as to avoid mistakenly using a sequence number over before its first use has been acknowledged. This alone does not guarantee that old duplicate data is drained from the net, so the sequence space has been made very large to reduce the probability that a wandering duplicate will cause trouble upon arrival. At 2 megabits/sec. it takes 4.5 hours to use up 2**32 octets of sequence space. Since the maximum segment lifetime in the net is not likely to exceed a few tens of seconds, this is deemed ample protection for foreseeable nets, even if data rates escalate to l0's of megabits/sec. At 100 megabits/sec, the cycle time is 5.4 minutes which may be a little short, but still within reason.

The basic duplicate detection and sequencing algorithm in TCP can be defeated, however, if a source TCP does not have any memory of the sequence numbers it last used on a given connection. For example, if the TCP were to start all connections with sequence number 0, then upon crashing and restarting, a TCP might re-form an earlier connection (possibly after half-open connection resolution) and emit packets with sequence numbers identical to or overlapping with packets still in the network which were emitted on an earlier incarnation of the same connection. In the absence of knowledge about the sequence numbers used on a particular connection, the TCP specification recommends that the source delay for MSL seconds before emitting segments on the connection, to allow time for segments from the earlier connection incarnation to drain from the system.

Even hosts which can remember the time of day and used it to select initial sequence number values are not immune from this problem (i.e., even if time of day is used to select an initial sequence number for each new connection incarnation).

Suppose, for example, that a connection is opened starting with sequence number S. Suppose that this connection is not used much and that eventually the initial sequence number function (ISN(t)) takes on a value equal to the sequence number, say S1, of the last segment sent by this TCP on a particular connection. Now suppose, at this instant, the host crashes, recovers, and establishes a new incarnation of the connection. The initial sequence number chosen is S1 = ISN(t) -- last used sequence number on old incarnation of connection! If the recovery occurs quickly enough, any old duplicates in the net bearing

sequence numbers in the neighborhood of S1 may arrive and be treated as new packets by the receiver of the new incarnation of the connection.

The problem is that the recovering host may not know for how long it crashed nor does it know whether there are still old duplicates in the system from earlier connection incarnations.

One way to deal with this problem is to deliberately delay emitting segments for one MSL after recovery from a crash- this is the "quite time" specification. Hosts which prefer to avoid waiting are willing to risk possible confusion of old and new packets at a given destination may choose not to wait for the "quite time". Implementors may provide TCP users with the ability to select on a connection by connection basis whether to wait after a crash, or may informally implement the "quite time" for all connections. Obviously, even where a user selects to "wait," this is not necessary after the host has been "up" for at least MSL seconds.

To summarize: every segment emitted occupies one or more sequence numbers in the sequence space, the numbers occupied by a segment are "busy" or "in use" until MSL seconds have passed, upon crashing a block of space-time is occupied by the octets of the last emitted segment, if a new connection is started too soon and uses any of the sequence numbers in the space-time footprint of the last segment of the previous connection incarnation, there is a potential sequence number overlap area which could cause confusion at the receiver.

Establishing a Connection

The "three-way handshake" is the procedure used to establish a connection. This procedure normally is initiated by one TCP and responded to by another TCP. The procedure also works if two TCP simultaneously initiate the procedure. When simultaneous attempt occurs, each TCP receives a "SYN" segment which carries no acknowledgment after it has sent a "SYN". Of course, the arrival of an old duplicate "SYN" segment can potentially make it appear, to the recipient, that a simultaneous connection initiation is in progress. Proper use of "reset" segments can disambiguate these cases.

Several examples of connection initiation follow. Although these examples do not show connection synchronization using data-carrying segments, this is perfectly legitimate, so long as the receiving TCP doesn't deliver the data to the user until it is clear the data is valid (i.e., the data must be buffered at the receiver until the connection reaches the ESTABLISHED state). The three-way handshake reduces the possibility of false connections. It is the implementation of a trade-off between memory and messages to provide information for this checking.

The simplest three-way handshake is shown in figure 7 below. The figures should be interpreted in the following way. Each line is numbered for reference purposes. Right arrows (-->) indicate departure of a TCP segment from TCP A to TCP B, or arrival of a segment at B from A. Left arrows (<--), indicate the reverse. Ellipsis (...) indicates a segment which is still in the network (delayed). An "XXX" indicates a segment which is lost or rejected. Comments appear in parentheses. TCP states represent the state AFTER the departure or arrival of the segment (whose contents are shown in the center of each line). Segment contents are shown in abbreviated form, with sequence number, control flags, and ACK field. Other fields such as window, addresses, lengths, and text have been left out in the interest of clarity.

RFC-793 Transmission Control Protocol - Establishing a Conneciton

Basic 3-Way Handshake for Connection Synchronization

	TCP A			TCP B
1	Closed			Listen
2	SYN-Sent	-->	SEQ: 100 CTL: SYN	--> SYN-Received
3	Established	<--	SEQ: 300 ACK: 100 CTL: SYN,ACK	<-- SYN-Received
4	Established	-->	SEQ: 101 ACK: 301 CTL: ACK	--> Established
5	Established	-->	SEQ: 101 ACK: 301 CTL: ACK data	--> Established

In line 2 of the above figure, TCP A begins by sending a SYN segment indicating that it will use sequence numbers starting with sequence number 100. In line 3, TCP B sends a SYN and acknowledges the SYN it received from TCP A. Note that the acknowledgment field indicates TCP B is now expecting to hear sequence 101, acknowledging the SYN which occupied sequence 100.

At line 4, TCP A responds with an empty segment containing an ACK for TCP B's SYN; and in line 5, TCP A sends some data. Note that the sequence number of the segment in line 5 is the same as in line 4 because the ACK does not occupy sequence number space (if it did, we would wind up ACKing ACK's!).

Simultaneous Connection Initiation

Simultaneous initiation is only slightly more complex, as is shown in figure 8. Each TCP cycles from CLOSED to SYN-SENT to SYN-RECEIVED to ESTABLISHED.

Simultaneous Connection Synchronization

	TCP A			TCP B
1	Closed			Listen
2	SYN-Sent	-->	SEQ: 100 CTL: SYN	...
3	SYN-Received	<--	SEQ: 300 CTL: SYN	<-- SYN-Sent
4		...	SEQ: 100 CTL: SYN	--> SYN-Received
5	SYN-Received	-->	SEQ: 100 ACK: 301 CTL: SYN,ACK	...
6	Established	<--	SEQ:300 ACK: 101 CTL: SYN,ACK	<-- SYN-Received
7		...	SEQ: 100 ACK: 301 CTL: SYN,ACK	--> Established

A TCP **must** support simultaneous open attempts.

> **Discussion**
> It sometimes surprises implementors that if two applications attempt to simultaneously connect to each other, only one connection is generated instead of two. This was an intentional design decision; don't try to "fix" it.

The principle reason for the three-way handshake is to prevent old duplicate connection initiations from causing confusion. To deal with this, a special control message, reset, has been devised. If the receiving TCP is in a non-synchronized state (i.e., SYN-SENT, SYN-RECEIVED), it returns to LISTEN on receiving an acceptable reset. If the TCP is in one of the synchronized states (ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT), it aborts the connection and informs its user. We discuss this latter case under "half-open" connections.

Recovery from Old Duplicate SYN

	TCP A			TCP B
1	Closed			Listen
2	SYN-Sent	-->	SEQ: 100 CTL: SYN	...
3	(duplicate)	-->	SEQ: 90 CTL: SYN	--> SYN-Received
4	SYN-Sent	<--	SEQ: 300 ACK: 91 CTL: SYN,ACK	<-- SYN-Received
5	SYN-Sent	-->	SEQ: 91 CTL: RST	--> Listen
6	Established	...	SEQ: 400 ACK: 101 CTL: SYN,ACK	--> SYN-Received
7	SYN-Sent	<--	SEQ: 400 ACK: 101 CTL: SYN,ACK	<-- SYN-Received
8	Established	-->	SEQ: 101 ACK: 401 CTL: ACK	--> Established

As a simple example of recovery from old duplicates, consider the figure above. At line 3, an old duplicate SYN arrives at TCP B. TCP B cannot tell that this is an old duplicate, so it responds normally (line 4). TCP A detects that the ACK field is incorrect and returns a RST (reset) with its SEQ field selected to make the segment believable. TCP B, on receiving the RST, returns to the LISTEN state. When the original SYN (pun intended) finally arrives at line 6, the synchronization proceeds normally. If the SYN at line 6 had arrived before the RST, a more complex exchange might have occurred with RST's sent in both directions.

RFC-793 Transmission Control Protocol - Establishing Connections

Half-Open Connections and Other Anomalies

An established connection is said to be "half-open" if one of the TCPs has closed or aborted the connection at its end without the knowledge of the other, or if the two ends of the connection have become desynchronized owing to a crash that resulted in loss of memory. Such connections will automatically become reset if an attempt is made to send data in either direction. However, half-open connections are expected to be unusual, and the recovery procedure is mildly involved.

If at site A the connection no longer exists, then an attempt by the user at site B to send any data on it will result in the site B TCP receiving a reset control message. Such a message indicates to the site B TCP that something is wrong, and it is expected to abort the connection.

Assume that two user processes A and B are communicating with one another when a crash occurs causing loss of memory to A's TCP. Depending on the operating system supporting A's TCP, it is likely that some error recovery mechanism exists. When the TCP is up again, A is likely to start again from the beginning or from a recovery point. As a result, A will probably try to OPEN the connection again or try to SEND on the connection it believes open. In the latter case, it receives the error message "connection not open" from the local (A's) TCP. In an attempt to establish the connection, A's TCP will send a segment containing SYN. This scenario leads to the example shown in figure 10. After TCP A crashes, the user attempts to re-open the connection. TCP B, in the meantime, thinks the connection is open.

	TCP A			**TCP B**
1	(CRASH)			S:300 R: 100
2	Closed			Established
3	SYN-Sent	--> SEQ: 400 CTL: SYN	-->	(??)
4	(!!)	<-- SEQ: 300 ACK: 100 CTL: ACK	<--	Established
5	SYN-Sent	--> SEQ: 100 CTL: RST	-->	(Abort!!)
6	SYN-Sent			Closed
7	SYN-Sent	--> SEQ: 400 CTL: SYN	-->	

Half-Open Connection Discovery

When the SYN arrives at line 3, TCP B, being in a synchronized state, and the incoming segment outside the window, responds with an acknowledgment indicating what sequence it next expects to hear (ACK 100). TCP A sees that this segment does not acknowledge anything it sent and, being unsynchronized, sends a reset (RST) because it has detected a half-open connection. TCP B aborts at line 5. TCP A will continue to try to establish the connection; the problem is now reduced to the basic 3-way handshake.

Half Open Connections - Alternative

An interesting alternative case occurs when TCP A crashes and TCP B tries to send data on what it thinks is a synchronized connection. This is illustrated in the figure below. In this case, the data arriving at TCP A from TCP B (line 2) is unacceptable because no such connection exists, so TCP A sends a RST. The RST is acceptable so TCP B processes it and aborts the connection.

Active Side Causes Half-Open Connection Discovery

	TCP A			TCP B
1	(CRASH)			S: 300 R: 100
2	(??)	<--	SEQ: 300 ACK: 100 Data: 10 CTL: ACK	<-- Established
3	SYN-Sent	-->	SEQ: 100 CTL: RST	--> (Abort!!)

In the figure below, we find the two TCPs A and B with passive connections waiting for SYN. An old duplicate arriving at TCP B (line 2) stirs B into action. A SYN-ACK is returned (line 3) and causes TCP A to generate a RST (the ACK in line 3 is not acceptable). TCP B accepts the reset and returns to its passive LISTEN state.

Old Duplicate SYN Initiates a Reset on two Passive Sockets

	TCP A			TCP B
1	Listen			Listen
2		...	SEQ: Z CTL: SYN	--> SYN-Received
3	(??)	<--	SEQ: X ACK: Z+1 CTL: SYN,ACK	<-- SYN-Received
4		-->	SEQ: Z+1 CTL: RST	--> return to Listen
5	Listen			Listen

A variety of other cases are possible, all of which are accounted for by the rules for RST generation and processing.

RFC-793 Transmission Control Protocol - Establishing Connections

Reset Generation

As a general rule, reset (RST) must be sent whenever a segment arrives which apparently is not intended for the current connection. A reset must not be sent if it is not clear that this is the case.

There are three groups of states:

1. If the connection does not exist (CLOSED) then a reset is sent in response to any incoming segment except another reset. In particular, SYNs addressed to a non-existent connection are rejected by this means.

   If the incoming segment has an ACK field, the reset takes its sequence number from the ACK field of the segment, otherwise the reset has sequence number zero and the ACK field is set to the sum of the sequence number and segment length of the incoming segment. The connection remains in the CLOSED state.

2. If the connection is in any non-synchronized state (LISTEN, SYN-SENT, SYN-RECEIVED), and the incoming segment acknowledges something not yet sent (the segment carries an unacceptable ACK), or if an incoming segment has a security level or compartment which does not exactly match the level and compartment requested for the connection, a reset is sent.

   If our SYN has not been acknowledged and the precedence level of the incoming segment is higher than the precedence level requested then either raise the local precedence level (if allowed by the user and the system) or send a reset; or if the precedence level of the incoming segment is lower than the precedence level requested then continue as if the precedence matched exactly (if the remote TCP cannot raise the precedence level to match ours this will be detected in the next segment it sends, and the connection will be terminated then). If our SYN has been acknowledged (perhaps in this incoming segment) the precedence level of the incoming segment must match the local precedence level exactly, if it does not a reset must be sent.

   If the incoming segment has an ACK field, the reset takes its sequence number from the ACK field of the segment, otherwise the reset has sequence number zero and the ACK field is set to the sum of the sequence number and segment length of the incoming segment. The connection remains in the same state.

3. If the connection is in a synchronized state (ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT), any unacceptable segment (out of window sequence number or unacceptible acknowledgment number) must elicit only an empty acknowledgment segment containing the current send-sequence number and an acknowledgment indicating the next sequence number expected to be received, and the connection remains in the same state.

   If an incoming segment has a security level, or compartment, or precedence which does not exactly match the level, and compartment, and precedence requested for the connection,a reset is sent and

connection goes to the CLOSED state. The reset takes its sequence number from the ACK field of the incoming segment.

Reset Processing

In all states except SYN-SENT, all reset (RST) segments are validated by checking their SEQ-fields. A reset is valid if its sequence number is in the window. In the SYN-SENT state (a RST received in response to an initial SYN), the RST is acceptable if the ACK field acknowledges the SYN.

The receiver of a RST first validates it, then changes state. If the receiver was in the LISTEN state, it ignores it. If the receiver was in SYN-RECEIVED state and had previously been in the LISTEN state, then the receiver returns to the LISTEN state, otherwise the receiver aborts the connection and goes to the CLOSED state. If the receiver was in any other state, it aborts the connection and advises the user and goes to the CLOSED state.

A TCP **should** allow a received RST segment to include data.

> **Discussion**
> It has been suggested that a RST segment could contain ASCII text that encoded and explained the cause of the RST. No standard has yet been established for such data.

RFC-793 Transmission Control Protocol - Functional Specification

Closing a Connection

CLOSE is an operation meaning "I have no more data to send." The notion of closing a full-duplex connection is subject to ambiguous interpretation, of course, since it may not be obvious how to treat the receiving side of the connection. We have chosen to treat CLOSE in a simplex fashion. The user who CLOSEs may continue to RECEIVE until he is told that the other side has CLOSED also. Thus, a program could initiate several SENDs followed by a CLOSE, and then continue to RECEIVE until signaled that a RECEIVE failed because the other side has CLOSED. We assume that the TCP will signal a user, even if no RECEIVEs are outstanding, that the other side has closed either normally or via an ABORT, so the user can terminate his side gracefully. A TCP will reliably deliver all buffers SENT before the connection was CLOSED so a user who expects no data in return need only wait to hear the connection was CLOSED successfully to know that all his data was received at the destination TCP. Users must keep reading connections they close for sending until the TCP says no more data.

The normal TCP close sequence delivers buffered data reliably in both directions. Since the two directions of a TCP connection are closed independently, it is possible for a connection to be "half closed", i.e., closed in only one direction, and a host is permitted to continue sending data in the open direction on a half-closed connection.

There are essentially three cases:

1) The user initiates by telling the TCP to CLOSE the connection
2) The remote TCP initiates by sending a FIN control signal
3) Both users CLOSE simultaneously

Case 1: Local user initiates the close

In this case, a FIN segment can be constructed and placed on the outgoing segment queue. No further SENDs from the user will be accepted by the TCP, and it enters the FIN-WAIT-1 state. RECEIVEs are allowed in this state. All segments preceding and including FIN will be retransmitted until acknowledged. When the other TCP has both acknowledged the FIN and sent a FIN of its own, the first TCP can ACK this FIN. Note that a TCP receiving a FIN will ACK but not send its own FIN until its user has CLOSED the connection also.

Case 2: TCP receives a FIN from the network

If an unsolicited FIN arrives from the network, the receiving TCP can ACK it and tell the user that the connection is closing. The user will respond with a CLOSE, upon which the TCP can send a FIN to the other TCP after sending any remaining data. The TCP then waits until its own FIN is acknowledged whereupon it deletes the connection. If an ACK is not forthcoming, after the user timeout the connection is aborted and the user is told.

Case 3: Both users close simultaneously

A simultaneous CLOSE by users at both ends of a connection causes FIN segments to be exchanged. When all segments preceding the FINs have been processed and acknowledged, each TCP can ACK the FIN it has received. Both will, upon receiving these ACKs, delete the connection.

RFC-793 Transmission Control Protocol - Closing a Connection

Normal Close Sequence

	TCP A			TCP B
1	Established			Established
2	(Close) FIN-Wait-1	-->	SEQ: 100 ACK: 300 CTL: FIN,ACK	--> Close-Wait
3	FIN-Wait-2	<--	SEQ: 300 ACK: 101 CTL: ACK	<-- Close-Wait
4	Time-Wait	<--	SEQ: 300 ACK: 101 CTL: FIN,ACK	<-- (Close) Last ACK
5	Time-Wait	-->	SEQ: 101 ACK: 301 CTL: ACK	--> Closed
6	(2 MSL) Closed			

RFC-793 Transmission Control Protocol - Closing a Connection

Simultaneous Close Sequence

	TCP A		TCP B
1	Estab		Estab
2	(Close) FIN-W-1	--> SEQ: 100 ACK: 300 CTL: FIN,ACK <-- SEQ: 300 ACK: 100 CTL: FIN,ACK SEQ: 100 ACK: 300 CTL: FIN, ACK	(Close) ... FIN-W-1 <-- ...
3	Closing	--> SEQ: 101 ACK: 301 CTL: ACK ... <-- SEQ: 301 ACK: 101 CTL: ACK <-- ... SEQ: 101 ACK: 301 CTL: ACK -->	Closing
4	Time-W (2 MSL) Closed		Time-W (2 MSL) Closed

Half-Duplex Close Sequence

A host **may** implement a "half-duplex" TCP close sequence, so that an application that has called CLOSE cannot continue to read data from the connection. If such a host issues a CLOSE call while received data is still pending in TCP, or if new data is received after CLOSE is called, its TCP **should** send a RST to show that data was lost.

When a connection is closed actively, it **must** linger in TIME-WAIT state for a time 2xMSL (Maximum Segment Lifetime). However, it **may** accept a new SYN from the remote TCP to reopen the connection directly from TIME-WAIT state, if it:

(1) assigns its initial sequence number for the new connection to be larger than the largest sequence number it used on the previous connection incarnation, and

(2) returns to TIME-WAIT state if the SYN turns out to be an old duplicate.

Discussion

TCP's full-duplex data-preserving close is a feature that is not included in the analogous ISO transport protocol TP4.

Some systems have not implemented half-closed connections, presumably because they do not fit into the I/O model of their particular operating system. On these systems, once an application has called CLOSE, it can no longer read input data from the connection; this is referred to as a "half-duplex" TCP close sequence.

The graceful close algorithm of TCP requires that the connection state remain defined on (at least) one end of the connection, for a timeout period of 2xMSL, i.e., 4 minutes. During this period, the (remote socket, local socket) pair that defines the connection is busy and cannot be reused. To shorten the time that a given port pair is tied up, some TCPs allow a new SYN to be accepted in TIME-WAIT state.

Precedence and Security

The intent is that connection be allowed only between ports operating with exactly the same security and compartment values and at the higher of the precedence level requested by the two ports.

The precedence and security parameters used in TCP are exactly those defined in the Internet Protocol (IP). Throughout this TCP specification the term "security/compartment" is intended to indicate the security parameters used in IP including security, compartment, user group, and handling restriction.

A connection attempt with mismatched security/compartment values or a lower precedence value must be rejected by sending a reset. Rejecting a connection due to too low a precedence only occurs after an acknowledgment of the SYN has been received.

Note that TCP modules which operate only at the default value of precedence will still have to check the precedence of incoming segments and possibly raise the precedence level they use on the connection.

The security paramaters may be used even in a non-secure environment (the values would indicate unclassified data), thus hosts in non-secure environments must be prepared to receive the security parameters, though they need not send them.

Data Communication

Once the connection is established data is communicated by the exchange of segments. Because segments may be lost due to errors (checksum test failure), or network congestion, TCP uses retransmission (after a timeout) to ensure delivery of every segment. Duplicate segments may arrive due to network or TCP retransmission. As discussed in the section on sequence numbers the TCP performs certain tests on the sequence and acknowledgment numbers in the segments to verify their acceptability.

The sender of data keeps track of the next sequence number to use in the variable SND.NXT. The receiver of data keeps track of the next sequence number to expect in the variable RCV.NXT. The sender of data keeps track of the oldest unacknowledged sequence number in the variable SND.UNA. If the data flow is momentarily idle and all data sent has been acknowledged then the three variables will be equal.

When the sender creates a segment and transmits it the sender advances SND.NXT. When the receiver accepts a segment it advances RCV.NXT and sends an acknowledgment. When the data sender receives an acknowledgment it advances SND.UNA. The extent to which the values of these variables differ is a measure of the delay in the communication. The amount by which the variables are advanced is the length of the data in the segment. Note that once in the ESTABLISHED state all segments must carry current acknowledgment information.

The CLOSE user call implies a push function, as does the FIN control flag in an incoming segment.

Efficiency Issues

Since RFC-793 was written, there has been extensive work on TCP algorithms to achieve efficient data communication. Later sections of this document describe required and recommended TCP algorithms to determine when to send data (When to Send Data), when to send an acknowledgment (When to Send an ACK Segment), and when to update the window (When to Send a Window Update).

Discussion

One important performance issue is "Silly Window Syndrome" or "SWS" [RFC-813], a stable pattern of small incremental window movements resulting in extremely poor TCP performance. Algorithms to avoid SWS are described for both the sending side (When to Send Data) and the receiving side (When to Send a Window Update). RFC-813 Window and Acknowledgement Strategy in TCP discusses several other issues as well.

In brief, SWS is caused by the receiver advancing the right window edge whenever it has any new buffer space available to receive data and by the sender using any incremental window, no matter how small, to send more data. The result can be a stable pattern of sending tiny data segments, even though both sender and receiver have a large total buffer space for the connection. SWS can only occur during the transmission of a large amount of data; if the connection goes quiescent, the problem will disappear. It is caused by typical straightforward implementation of window management, but the sender and receiver algorithms given below will avoid it.

Another important TCP performance issue is that some applications, especially remote login to character-at- a-time hosts, tend to send streams of one-octet data segments. To avoid deadlocks, every TCP SEND call from such applications must be "pushed", either explicitly by the application or else implicitly by TCP. The result may be a stream of TCP segments that contain one data octet each, which makes very inefficient use of the Internet and contributes to Internet congestion. The Nagle Algorithm described in When to Send Data provides a simple and effective solution to this problem. It does have the effect of clumping characters over Telnet connections; this may initially surprise users accustomed to single-character echo, but user acceptance has not been a problem.

Note that the Nagle algorithm and the send SWS avoidance algorithm play complementary roles in improving performance. The Nagle algorithm discourages sending tiny segments when the data to be sent increases in small increments, while the SWS avoidance algorithm discourages small segments resulting from the right window edge advancing in small increments.

A careless implementation can send two or more acknowledgment segments per data segment received. For example, suppose the receiver acknowledges every data segment immediately. When the application program subsequently consumes the data and increases the available receive buffer space again, the receiver may send a second acknowledgment segment to update the window at the sender. The extreme case occurs with single-character segments on TCP connections using the Telnet protocol for remote login service. Some implementations have been observed in which each incoming 1-character segment generates three return segments: (1) the acknowledgment, (2) a one byte increase in the window, and (3) the echoed

character, respectively.

Retransmission Timeout

Because of the variability of the networks that compose an internetwork system and the wide range of uses of TCP connections the retransmission timeout must be dynamically determined.

A host TCP **must** implement Karn's algorithm and Jacobson's algorithm for computing the retransmission timeout ("RTO").

- Jacobson's algorithm for computing the smoothed round- trip ("RTT") time incorporates a simple measure of the variance [TCP:7].
- Karn's algorithm for selecting RTT measurements ensures that ambiguous round-trip times will not corrupt the calculation of the smoothed round-trip time [TCP:6].

This implementation also **must** include "exponential backoff" for successive RTO values for the same segment. Retransmission of SYN segments **should** use the same algorithm as data segments.

The following values **should** be used to initialize the estimation parameters for a new connection:

(a) RTT = 0 seconds.

(b) RTO = 3 seconds. (The smoothed variance is to be initialized to the value that will result in this RTO).

The recommended upper and lower bounds on the RTO are known to be inadequate on large internets. The lower bound **should** be measured in fractions of a second (to accommodate high speed LANs) and the upper bound should be 2*MSL, i.e., 240 seconds.

Discussion
Experience has shown that these initialization values are reasonable, and that in any case the Karn and Jacobson algorithms make TCP behavior reasonably insensitive to the initial parameter choices.

If a retransmitted packet is identical to the original packet (which implies not only that the data boundaries have not changed, but also that the window and acknowledgment fields of the header have not changed), then the same IP Identification field **may** be used.

Implementation
Some TCP implementors have chosen to "packetize" the data stream, i.e., to pick segment boundaries when segments are originally sent and to queue these segments in a "retransmission queue" until they are acknowledged. Another design (which may be simpler) is to defer packetizing until each time data is transmitted or retransmitted, so there will be no segment retransmission queue.

In an implementation with a segment retransmission queue, TCP performance may be enhanced by repacketizing the segments awaiting acknowledgment when the first retransmission timeout occurs. That is, the outstanding segments that fitted would be combined into one maximum-sized segment, with a new IP Identification value. The TCP would then retain this combined segment in the retransmit queue until it was acknowledged. However, if the first two segments in the retransmission queue totalled more than one maximum- sized segment, the TCP would retransmit only the first segment using the original IP Identification field.

The Communication of Urgent Information

The objective of the TCP urgent mechanism is to allow the sending user to stimulate the receiving user to accept some urgent data and to permit the receiving TCP to indicate to the receiving user when all the currently known urgent data has been received by the user.

This mechanism permits a point in the data stream to be designated as the end of urgent information. Whenever this point is in advance of the receive sequence number (RCV.NXT) at the receiving TCP, that TCP must tell the user to go into "urgent mode"; when the receive sequence number catches up to the urgent pointer, the TCP must tell user to go into "normal mode". If the urgent pointer is updated while the user is in "urgent mode", the update will be invisible to the user.

The method employs a urgent field which is carried in all segments transmitted. The URG control flag indicates that the urgent field is meaningful and must be added to the segment sequence number to yield the urgent pointer. The absence of this flag indicates that there is no urgent data outstanding.

To send an urgent indication the user must also send at least one data octet. If the sending user also indicates a push, timely delivery of the urgent information to the destination process is enhanced.

Managing the Window

The window sent in each segment indicates the range of sequence numbers the sender of the window (the data receiver) is currently prepared to accept. There is an assumption that this is related to the currently available data buffer space available for this connection.

Indicating a large window encourages transmissions. If more data arrives than can be accepted, it will be discarded. This will result in excessive retransmissions, adding unnecessarily to the load on the network and the TCPs. Indicating a small window may restrict the transmission of data to the point of introducing a round trip delay between each new segment transmitted.

The mechanisms provided allow a TCP to advertise a large window and to subsequently advertise a much smaller window without having accepted that much data. This, so called "shrinking the window", is strongly discouraged. The robustness principle dictates that TCPs will not shrink the window themselves, but will be prepared for such behavior on the part of other TCPs.

The sending TCP must be prepared to accept from the user and send at least one octet of new data even if the send window is zero. The sending TCP must regularly retransmit to the receiving TCP even when the window is zero. Two minutes is recommended for the retransmission interval when the window is zero. This retransmission is essential to guarantee that when either TCP has a zero window the re-opening of the window will be reliably reported to the other.

When the receiving TCP has a zero window and a segment arrives it must still send an acknowledgment showing its next expected sequence number and current window (zero).

The sending TCP packages the data to be transmitted into segments which fit the current window, and may repackage segments on the retransmission queue. Such repackaging is not required, but may be helpful.

In a connection with a one-way data flow, the window information will be carried in acknowledgment segments that all have the same sequence number so there will be no way to reorder them if they arrive out of order. This is not a serious problem, but it will allow the window information to be on occasion temporarily based on old reports from the data receiver. A refinement to avoid this problem is to act on the window information from segments that carry the highest acknowledgment number (that is segments with acknowledgment number equal or greater than the highest previously received).

Shrinking the Window

A TCP receiver **should** NOT shrink the window, i.e., move the right window edge to the left. However, a sending TCP **must** be robust against window shrinking, which may cause the "useable window" (see When to Send Data) to become negative.

If this happens, the sender **should** NOT send new data, but **should** retransmit normally the old unacknowledged data between SND.UNA and SND.UNA+SND.WND. The sender **may** also retransmit old data beyond SND.UNA+SND.WND, but **should** NOT time out the connection if data beyond the right window edge is not acknowledged. If the window shrinks to zero, the TCP **must** probe it in the standard way.

> **Discussion**
> Many TCP implementations become confused if the window shrinks from the right after data has been sent into a larger window. Note that TCP has a heuristic to select the latest window update despite possible datagram reordering; as a result, it may ignore a window update with a smaller window than previously offered if neither the sequence number nor the acknowledgment number is increased.

Probing Zero Windows

Probing of zero (offered) windows **must** be supported.

A TCP **may** keep its offered receive window closed indefinitely. As long as the receiving TCP continues to send acknowledgments in response to the probe segments, the sending TCP **must** allow the connection to stay open.

> **Discussion**
> It is extremely important to remember that ACK (acknowledgment) segments that contain no data are not reliably transmitted by TCP. If zero window probing is not supported, a connection may hang forever when an ACK segment that re-opens the window is lost.
>
> The delay in opening a zero window generally occurs when the receiving application stops taking data from its TCP. For example, consider a printer daemon application, stopped because the printer ran out of paper.

The transmitting host **should** send the first zero-window probe when a zero window has existed for the retransmission timeout period, and **should** increase exponentially the interval between successive probes.

> **Discussion**
> This procedure minimizes delay if the zero-window condition is due to a lost ACK segment containing a window-opening update. Exponential backoff is recommended, possibly with some maximum interval not specified here. This procedure is similar to that of the retransmission algorithm, and it may be possible to combine the two procedures in the implementation.

Window Management Suggestions

The window management procedure has significant influence on the communication performance. The following comments are suggestions to implementers.

Allocating a very small window causes data to be transmitted in many small segments when better performance is achieved using fewer large segments.

One suggestion for avoiding small windows is for the receiver to defer updating a window until the additional allocation is at least X percent of the maximum allocation possible for the connection (where X might be 20 to 40).

Another suggestion is for the sender to avoid sending small segments by waiting until the window is large enough before sending data. If the the user signals a push function then the data must be sent even if it is a small segment.

Note that the acknowledgments should not be delayed or unnecessary retransmissions will result. One strategy would be to send an acknowledgment when a small segment arrives (with out updating the window information), and then to send another acknowledgment with new window information when the window is larger.

The segment sent to probe a zero window may also begin a break up of transmitted data into smaller and smaller segments. If a segment containing a single data octet sent to probe a zero window is accepted, it consumes one octet of the window now available. If the sending TCP simply sends as much as it can whenever the window is non zero, the transmitted data will be broken into alternating big and small segments. As time goes on, occasional pauses in the receiver making window allocation available will result in breaking the big segments into a small and not quite so big pair. And after a while the data transmission will be in mostly small segments.

The suggestion here is that the TCP implementations need to actively attempt to combine small window allocations into larger windows, since the mechanisms for managing the window tend to lead to many small windows in the simplest minded implementations.

Interfaces

There are of course two interfaces of concern: the user/TCP interface and the TCP/lower-level interface. We have a fairly elaborate model of the user/TCP interface, but the interface to the lower level protocol module is left unspecified here, since it will be specified in detail by the specification of the lowel level protocol. For the case that the lower level is IP we note some of the parameter values that TCPs might use.

User/TCP Interface

The following functional description of user commands to the TCP is, at best, fictional, since every operating system will have different facilities. Consequently, we must warn readers that different TCP implementations may have different user interfaces. However, all TCPs must provide a certain minimum set of services to guarantee that all TCP implementations can support the same protocol hierarchy. This section specifies the functional interfaces required of all TCP implementations.

TCP User Commands

The following sections functionally characterize a USER/TCP interface. The notation used is similar to most procedure or function calls in high level languages, but this usage is not meant to rule out trap type service calls (e.g., SVCs, UUOs, EMTs).

The user commands described below specify the basic functions the TCP must perform to support interprocess communication. Individual implementations must define their own exact format, and may provide combinations or subsets of the basic functions in single calls. In particular, some implementations may wish to automatically OPEN a connection on the first SEND or RECEIVE issued by the user for a given connection.

In providing interprocess communication facilities, the TCP must not only accept commands, but must also return information to the processes it serves. The latter consists of:

(a) general information about a connection (e.g., interrupts, remote close, binding of unspecified foreign socket).

(b) replies to specific user commands indicating success or various types of failure.

RFC-793 Transmission Control Protocol - User/TCP Interfaces

Open

Format: OPEN (local port, foreign socket, active/passive [, timeout] [, precedence] [, security/compartment] [, options]) -> local connection name

We assume that the local TCP is aware of the identity of the processes it serves and will check the authority of the process to use the connection specified. Depending upon the implementation of the TCP, the local network and TCP identifiers for the source address will either be supplied by the TCP or the lower level protocol (e.g., IP). These considerations are the result of concern about security, to the extent that no TCP be able to masquerade as another one, and so on. Similarly, no process can masquerade as another without the collusion of the TCP.

If the active/passive flag is set to passive, then this is a call to LISTEN for an incoming connection. A passive open may have either a fully specified foreign socket to wait for a particular connection or an unspecified foreign socket to wait for any call. A fully specified passive call can be made active by the subsequent execution of a SEND.

A transmission control block (TCB) is created and partially filled in with data from the OPEN command parameters.

On an active OPEN command, the TCP will begin the procedure to synchronize (i.e., establish) the connection at once.

The timeout, if present, permits the caller to set up a timeout for all data submitted to TCP. If data is not successfully delivered to the destination within the timeout period, the TCP will abort the connection. The present global default is five minutes.

The TCP or some component of the operating system will verify the users authority to open a connection with the specified precedence or security/compartment. The absence of precedence or security/compartment specification in the OPEN call indicates the default values must be used.

TCP will accept incoming requests as matching only if the security/compartment information is exactly the same and only if the precedence is equal to or higher than the precedence requested in the OPEN call.

The precedence for the connection is the higher of the values requested in the OPEN call and received from the incoming request, and fixed at that value for the life of the connection.Implementers may want to give the user control of this precedence negotiation. For example, the user might be allowed to specify that the precedence must be exactly matched, or that any attempt to raise the precedence be confirmed by the user.

A local connection name will be returned to the user by the TCP. The local connection name can then be used as a short hand term for the connection defined by the <local socket, foreign socket> pair.

Passive Open Calls

Every passive OPEN call either creates a new connection record in LISTEN state, or it returns an error; it **must not** affect any previously created connection record.

A TCP that supports multiple concurrent users **must** provide an OPEN call that will functionally allow an application to LISTEN on a port while a connection block with the same local port is in SYN-SENT or SYN-RECEIVED state.

> **Discussion**
> Some applications (e.g., SMTP servers) may need to handle multiple connection attempts at about the same time. The probability of a connection attempt failing is reduced by giving the application some means of listening for a new connection at the same time that an earlier connection attempt is going through the three- way handshake.
>
> **Implementation**
> Acceptable implementations of concurrent opens may permit multiple passive OPEN calls, or they may allow "cloning" of LISTEN-state connections from a single passive OPEN call.

Send

Format: SEND (local connection name, buffer address, byte count, PUSH flag, URGENT flag [,timeout])

This call causes the data contained in the indicated user buffer to be sent on the indicated connection. If the connection has not been opened, the SEND is considered an error. Some implementations may allow users to SEND first; in which case, an automatic OPEN would be done. If the calling process is not authorized to use this connection, an error is returned.

If the PUSH flag is set, the data must be transmitted promptly to the receiver, and the PUSH bit will be set in the last TCP segment created from the buffer. If the PUSH flag is not set, the data may be combined with data from subsequent SENDs for transmission efficiency.

If the URGENT flag is set, segments sent to the destination TCP will have the urgent pointer set. The receiving TCP will signal the urgent condition to the receiving process if the urgent pointer indicates that data preceding the urgent pointer has not been consumed by the receiving process. The purpose of urgent is to stimulate the receiver to process the urgent data and to indicate to the receiver when all the currently known urgent data has been received. The number of times the sending user's TCP signals urgent will not necessarily be equal to the number of times the receiving user will be notified of the presence of urgent data.

If no foreign socket was specified in the OPEN, but the connection is established (e.g., because a LISTENing connection has become specific due to a foreign segment arriving for the local socket), then the designated buffer is sent to the implied foreign socket. Users who make use of OPEN with an unspecified foreign socket can make use of SEND without ever explicitly knowing the foreign socket address.

However, if a SEND is attempted before the foreign socket becomes specified, an error will be returned. Users can use the STATUS call to determine the status of the connection. In some implementations the TCP may notify the user when an unspecified socket is bound.

If a timeout is specified, the current user timeout for this connection is changed to the new one.

In the simplest implementation, SEND would not return control to the sending process until either the transmission was complete or the timeout had been exceeded. However, this simple method is both subject to deadlocks (for example, both sides of the connection might try to do SENDs before doing any RECEIVEs) and offers poor performance, so it is not recommended. A more sophisticated implementation would return immediately to allow the process to run concurrently with network I/O, and, furthermore, to allow multiple SENDs to be in progress. Multiple SENDs are served in first come, first served order, so the TCP will queue those it cannot service immediately.

We have implicitly assumed an asynchronous user interface in which a SEND later elicits some kind of SIGNAL or pseudo-interrupt from the serving TCP. An alternative is to return a response immediately. For instance, SENDs might return immediate local acknowledgment, even if the segment sent had not been acknowledged by the distant TCP. We could optimistically assume eventual success. If we are wrong, the connection will close anyway due to the timeout. In implementations of this kind (synchronous), there will still be some asynchronous signals, but these will deal with the connection itself, and not with specific segments or buffers.

In order for the process to distinguish among error or success indications for different SENDs, it might be appropriate for thebuffer address to be returned along with the coded

response to the SEND request. TCP-to-user signals are discussed below, indicating the information which should be returned to the calling process.

Receive

Format: RECEIVE (local connection name, buffer address, byte count) -> byte count, urgent flag, push flag

This command allocates a receiving buffer associated with the specified connection. If no OPEN precedes this command or the calling process is not authorized to use this connection, an error is returned.

In the simplest implementation, control would not return to the calling program until either the buffer was filled, or some error occurred, but this scheme is highly subject to deadlocks. A more sophisticated implementation would permit several RECEIVEs to be outstanding at once. These would be filled as segments arrive. This strategy permits increased throughput at the cost of a more elaborate scheme (possibly asynchronous) to notify the calling program that a PUSH has been seen or a buffer filled.

If enough data arrive to fill the buffer before a PUSH is seen, the PUSH flag will not be set in the response to the RECEIVE. The buffer will be filled with as much data as it can hold. If a PUSH is seen before the buffer is filled the buffer will be returned partially filled and **(optionally)** PUSH indicated.

If there is urgent data the user will have been informed as soon as it arrived via a TCP-to-user signal. The receiving user should thus be in "urgent mode". If the URGENT flag is on, additional urgent data remains. If the URGENT flag is off, this call to RECEIVE has returned all the urgent data, and the user may now leave "urgent mode". Note that data following the urgent pointer (non-urgent data) cannot be delivered to the user in the same buffer with preceeding urgent data unless the boundary is clearly marked for the user.

To distinguish among several outstanding RECEIVEs and to take care of the case that a buffer is not completely filled, the return code is accompanied by both a buffer pointer and a byte count indicating the actual length of the data received.

Alternative implementations of RECEIVE might have the TCP allocate buffer storage, or the TCP might share a ring buffer with the user.

Close

Format: CLOSE (local connection name)

This command causes the connection specified to be closed. If the connection is not open or the calling process is not authorized to use this connection, an error is returned. Closing connections is intended to be a graceful operation in the sense that outstanding SENDs will be transmitted (and retransmitted), as flow control permits, until all have been serviced. Thus, it should be acceptable to make several SEND calls, followed by a CLOSE, and expect all the data to be sent to the destination. It should also be clear that users should continue to RECEIVE on CLOSING connections, since the other side may be trying to transmit the last of its data. Thus, CLOSE means "I have no more to send" but does not mean "I will not receive any more." It may happen (if the user level protocol is not well thought out) that the closing side is unable to get rid of all its data before timing out. In this event, CLOSE turns into ABORT, and the closing TCP gives up.

The user may CLOSE the connection at any time on his own initiative, or in response to various prompts from the TCP (e.g., remote close executed, transmission timeout exceeded, destination inaccessible).

Because closing a connection requires communication with the foreign TCP, connections may remain in the closing state for a short time. Attempts to reopen the connection before the TCP replies to the CLOSE command will result in error responses.

Close also implies push function.

Status

Format: STATUS (local connection name) -> status data

This is an implementation dependent user command and could be excluded without adverse effect. Information returned would typically come from the TCB associated with the connection.

This command returns a data block containing the following information:

local socket,
foreign socket,
local connection name,
receive window,
send window,
connection state,
number of buffers awaiting acknowledgment,
number of buffers pending receipt,
urgent state,
precedence,
security/compartment,
and transmission timeout.

Depending on the state of the connection, or on the implementation itself, some of this information may not be available or meaningful. If the calling process is not authorized to use this connection, an error is returned. This prevents unauthorized processes from gaining information about a connection.

Abort

Format: ABORT (local connection name)

This command causes all pending SENDs and RECEIVES to be aborted, the TCB to be removed, and a special RESET message to be sent to the TCP on the other side of the connection. Depending on the implementation, users may receive abort indications for each outstanding SEND or RECEIVE, or may simply receive an ABORT-acknowledgment.

TCP-to-User Messages

It is assumed that the operating system environment provides a means for the TCP to asynchronously signal the user program. When the TCP does signal a user program, certain information is passed to the user. Often in the specification the information will be an error message. In other cases there will be information relating to the completion of processing a SEND or RECEIVE or other user call.

The following information is provided:

Local Connection Name	Always
Response String	Always
Buffer Address	Send & Receive
Byte count (counts bytes received)	Receive
Push flag	Receive **(Optional)**
Urgent flag	Receive

TCP/Lower-Level Interface

The TCP calls on a lower level protocol module to actually send and receive information over a network. One case is that of the ARPA internetwork system where the lower level module is the Internet Protocol (IP) [2].

If the lower level protocol is IP it provides arguments for a type of service and for a time to live. TCP uses the following settings for these parameters:

Type of Service	=	Precedence: routine, Delay: normal, Throughput: normal, Reliability: normal; or 00000000.
Time to Live	=	Configurable, see IP Time-to-Live for Discussion

> Note that the assumed maximum segment lifetime is two minutes. Here we explicitly ask that a segment be destroyed if it cannot be delivered by the internet system within one minute.

If the lower level is IP (or other protocol that provides this feature) and source routing is used, the interface must allow the route information to be communicated. This is especially important so that the source and destination addresses used in the TCP checksum be the originating source and ultimate destination. It is also important to preserve the return route to answer connection requests.

Any lower level protocol will have to provide the source address, destination address, and protocol fields, and some way to determine the "TCP length", both to provide the functional equivlent service of IP and to be used in the TCP checksum.

Event Processing

The processing depicted in this section is an example of one possible implementation. Other implementations may have slightly different processing sequences, but they should differ from those in this section only in detail, not in substance.

The activity of the TCP can be characterized as responding to events. The events that occur can be cast into three categories: user calls, arriving segments, and timeouts. This section describes the processing the TCP does in response to each of the events. In many cases the processing required depends on the state of the connection.

Events that occur:

User Calls

OPEN
SEND
RECEIVE
CLOSE
ABORT
STATUS

Arriving Segments

SEGMENT ARRIVES

Timeouts

USER TIMEOUT
RETRANSMISSION TIMEOUT
TIME-WAIT TIMEOUT

The model of the TCP/user interface is that user commands receive an immediate return and possibly a delayed response via an event or pseudo interrupt. In the following descriptions, the term "signal" means cause a delayed response.

Error responses are given as character strings. For example, user commands referencing connections that do not exist receive "error: connection not open".

Please note in the following that all arithmetic on sequence numbers, acknowledgment numbers, windows, et cetera, is modulo 2**32 the size of the sequence number space. Also note that "=<" means less than or equal to (modulo 2**32).

A natural way to think about processing incoming segments is to imagine that they are first tested for proper sequence number (i.e., that their contents lie in the range of the expected "receive window" in the sequence number space) and then that they are generally queued and processed in sequence number order.

When a segment overlaps other already received segments we reconstruct the segment to contain just the new data, and adjust the header fields to be consistent.

Note that if no state change is mentioned the TCP stays in the same state.

Processing Queued Segments

While it is not strictly required, a TCP **should** be capable of queueing out-of-order TCP segments. Change the "may" in the last sentence of the first paragraph on page 70 to "should".

> **Discussion**
> Some small-host implementations have omitted segment queueing because of limited buffer space. This omission may be expected to adversely affect TCP throughput, since loss of a single segment causes all later segments to appear to be "out of sequence".

In general, the processing of received segments **must** be implemented to aggregate ACK segments whenever possible. For example, if the TCP is processing a series of queued segments, it **must** process them all before sending any ACK segments.

Acknowledging Queued Segments
A TCP **may** send an ACK segment acknowledging RCV.NXT when a valid segment arrives that is in the window but not at the left window edge.

> **Discussion**
> RFC-793 was originally ambiguous about whether or not an ACK segment should be sent when an out-of-order segment was received, i.e., when SEG.SEQ was unequal to RCV.NXT.
>
> One reason for ACKing out-of-order segments might be to support an experimental algorithm known as "fast retransmit". With this algorithm, the sender uses the "redundant" ACK's to deduce that a segment has been lost before the retransmission timer has expired. It counts the number of times an ACK has been received with the same value of SEG.ACK and with the same right window edge. If more than a threshold number of such ACK's is received, then the segment containing the octets starting at SEG.ACK is assumed to have been lost and is retransmitted, without awaiting a timeout. The threshold is chosen to compensate for the maximum likely segment reordering in the Internet. There is not yet enough experience with the fast retransmit algorithm to determine how useful it is.

RFC-793 Transmission Control Protocol - Event Processing

OPEN Call

CLOSED STATE (i.e., TCB does not exist)

Create a new transmission control block (TCB) to hold connection state information. Fill in local socket identifier, foreign socket, precedence, security/compartment, and user timeout information. Note that some parts of the foreign socket may be unspecified in a passive OPEN and are to be filled in by the parameters of the incoming SYN segment. Verify the security and precedence requested are allowed for this user, if not return "error: precedence not allowed" or "error: security/compartment not allowed." If passive enter the LISTEN state and return. If active and the foreign socket is unspecified, return "error: foreign socket unspecified"; if active and the foreign socket is specified, issue a SYN segment. An initial send sequence number (ISS) is selected. A SYN segment of the form <SEQ=ISS><CTL=SYN> is sent. Set SND.UNA to ISS, SND.NXT to ISS+1, enter SYN-SENT state, and return.

If the caller does not have access to the local socket specified, return "error: connection illegal for this process". If there is no room to create a new connection, return "error: insufficient resources".

LISTEN STATE

If active and the foreign socket is specified, then change the connection from passive to active, select an ISS. Send a SYN segment, set SND.UNA to ISS, SND.NXT to ISS+1. Enter SYN-SENT state. Data associated with SEND may be sent with SYN segment or queued for transmission after entering ESTABLISHED state. The urgent bit if requested in the command must be sent with the data segments sent as a result of this command. If there is no room to queue the request, respond with "error: insufficient resources". If Foreign socket was not specified, then return "error: foreign socket unspecified".

SYN-SENT STATE
SYN-RECEIVED STATE
ESTABLISHED STATE
FIN-WAIT-1 STATE
FIN-WAIT-2 STATE
CLOSE-WAIT STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

Return "error: connection already exists".

RFC-793 Transmission Control Protocol - Event Processing

SEND Call

CLOSED STATE (i.e., TCB does not exist)

If the user does not have access to such a connection, then return "error: connection illegal for this process".

Otherwise, return "error: connection does not exist".

LISTEN STATE

If the foreign socket is specified, then change the connection from passive to active, select an ISS. Send a SYN segment, set SND.UNA to ISS, SND.NXT to ISS+1. Enter SYN-SENT state. Data associated with SEND may be sent with SYN segment or queued for transmission after entering ESTABLISHED state. The urgent bit if requested in the command must be sent with the data segments sent as a result of this command. If there is no room to queue the request, respond with "error: insufficient resources". If Foreign socket was not specified, then return "error: foreign socket unspecified".

SYN-SENT STATE
SYN-RECEIVED STATE

Queue the data for transmission after entering ESTABLISHED state. If no space to queue, respond with "error: insufficient resources".

ESTABLISHED STATE
CLOSE-WAIT STATE

Segmentize the buffer and send it with a piggybacked acknowledgment (acknowledgment value = RCV.NXT). If there is insufficient space to remember this buffer, simply return "error: insufficient resources".

If the urgent flag is set, then SND.UP <- SND.NXT-1 and set the urgent pointer in the outgoing segments.

FIN-WAIT-1 STATE
FIN-WAIT-2 STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

Return "error: connection closing" and do not service request.

RECEIVE Call

CLOSED STATE (i.e., TCB does not exist)

> If the user does not have access to such a connection, return "error: connection illegal for this process".
>
> Otherwise return "error: connection does not exist".

LISTEN STATE
SYN-SENT STATE
SYN-RECEIVED STATE

> Queue for processing after entering ESTABLISHED state. If there is no room to queue this request, respond with "error: insufficient resources".

ESTABLISHED STATE
FIN-WAIT-1 STATE
FIN-WAIT-2 STATE

> If insufficient incoming segments are queued to satisfy the request, queue the request. If there is no queue space to remember the RECEIVE, respond with "error: insufficient resources".
>
> Reassemble queued incoming segments into receive buffer and return to user. Mark "push seen" (PUSH) if this is the case.
>
> If RCV.UP is in advance of the data currently being passed to the user notify the user of the presence of urgent data.
>
> When the TCP takes responsibility for delivering data to the user that fact must be communicated to the sender via an acknowledgment. The formation of such an acknowledgment is described below in the discussion of processing an incoming segment.

CLOSE-WAIT STATE

> Since the remote side has already sent FIN, RECEIVEs must be satisfied by text already on hand, but not yet delivered to the user. If no text is awaiting delivery, the RECEIVE will get a "error: connection closing" response. Otherwise, any remaining text can be used to satisfy the RECEIVE.

CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

> Return "error: connection closing".

RFC-793 Transmission Control Protocol - Event Processing

CLOSE Call

CLOSED STATE (i.e., TCB does not exist)

> If the user does not have access to such a connection, return "error: connection illegal for this process".
>
> Otherwise, return "error: connection does not exist".

LISTEN STATE

> Any outstanding RECEIVEs are returned with "error: closing" responses. Delete TCB, enter CLOSED state, and return.

SYN-SENT STATE

> Delete the TCB and return "error: closing" responses to any queued SENDs, or RECEIVEs.

SYN-RECEIVED STATE

> If no SENDs have been issued and there is no pending data to send, then form a FIN segment and send it, and enter FIN-WAIT-1 state; otherwise queue for processing after entering ESTABLISHED state.

ESTABLISHED STATE

> Queue this until all preceding SENDs have been segmentized, then form a FIN segment and send it. In any case, enter FIN-WAIT-1 state.

FIN-WAIT-1 STATE
FIN-WAIT-2 STATE

> Strictly speaking, this is an error and should receive a "error: connection closing" response. An "ok" response would be acceptable, too, as long as a second FIN is not emitted (the first FIN may be retransmitted though).

CLOSE-WAIT STATE

> Queue this request until all preceding SENDs have been segmentized; then send a FIN segment, enter LAST-ACK state.

CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

> Respond with "error: connection closing".

ABORT Call

CLOSED STATE (i.e., TCB does not exist)

> If the user should not have access to such a connection, return "error: connection illegal for this process".
>
> Otherwise return "error: connection does not exist".

LISTEN STATE

> Any outstanding RECEIVEs should be returned with "error: connection reset" responses. Delete TCB, enter CLOSED state, and return.

SYN-SENT STATE

> All queued SENDs and RECEIVEs should be given "connection reset" notification, delete the TCB, enter CLOSED state, and return.

SYN-RECEIVED STATE
ESTABLISHED STATE
FIN-WAIT-1 STATE
FIN-WAIT-2 STATE
CLOSE-WAIT STATE

> Send a reset segment:
>
> <SEQ=SND.NXT><CTL=RST>
>
> All queued SENDs and RECEIVEs should be given "connection reset" notification; all segments queued for transmission (except for the RST formed above) or retransmission should be flushed, delete the TCB, enter CLOSED state, and return.

CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

> Respond with "ok" and delete the TCB, enter CLOSED state, and return.

STATUS Call

CLOSED STATE (i.e., TCB does not exist)

If the user should not have access to such a connection, return "error: connection illegal for this process".

Otherwise return "error: connection does not exist".

LISTEN STATE

Return "state = LISTEN", and the TCB pointer.

SYN-SENT STATE

Return "state = SYN-SENT", and the TCB pointer.

SYN-RECEIVED STATE

Return "state = SYN-RECEIVED", and the TCB pointer.

ESTABLISHED STATE

Return "state = ESTABLISHED", and the TCB pointer.

FIN-WAIT-1 STATE

Return "state = FIN-WAIT-1", and the TCB pointer.

FIN-WAIT-2 STATE

Return "state = FIN-WAIT-2", and the TCB pointer.

CLOSE-WAIT STATE

Return "state = CLOSE-WAIT", and the TCB pointer.

CLOSING STATE

Return "state = CLOSING", and the TCB pointer.

LAST-ACK STATE

Return "state = LAST-ACK", and the TCB pointer.

TIME-WAIT STATE

Return "state = TIME-WAIT", and the TCB pointer.

RFC-793 Transmission Control Protocol - Event Processing

SEGMENT ARRIVES

If the state is CLOSED (i.e., TCB does not exist) then

all data in the incoming segment is discarded. An incoming segment containing a RST is discarded. An incoming segment not containing a RST causes a RST to be sent in response. The acknowledgment and sequence field values are selected to make the reset sequence acceptable to the TCP that sent the offending segment.

If the ACK bit is off, sequence number zero is used,

<SEQ=0><ACK=SEG.SEQ+SEG.LEN><CTL=RST,ACK>

If the ACK bit is on,

<SEQ=SEG.ACK><CTL=RST>

Return.

If the state is LISTEN then

first check for an RST

An incoming RST should be ignored. Return.

second check for an ACK

Any acknowledgment is bad if it arrives on a connection still in the LISTEN state. An acceptable reset segment should be formed for any arriving ACK-bearing segment. The RST should be formatted as follows:

<SEQ=SEG.ACK><CTL=RST>

Return.

third check for a SYN

If the SYN bit is set, check the security. If the security/compartment on the incoming segment does not exactly match the security/compartment in the TCB then send a reset and return.

<SEQ=0><ACK=SEG.SEQ+SEG.LEN><CTL=RST,ACK>

If the SEG.PRC is greater than the TCB.PRC then if allowed by the user and the system set TCB.PRC<-SEG.PRC, if not allowed send a reset and return.

<SEQ=SEG.ACK><CTL=RST>

If the SEG.PRC is less than the TCB.PRC then continue.

Set RCV.NXT to SEG.SEQ+1, IRS is set to SEG.SEQ and any other control or text should be queued for processing later. ISS should be selected and a SYN segment sent of the form:

<SEQ=ISS><ACK=RCV.NXT><CTL=SYN,ACK>

SND.NXT is set to ISS+1 and SND.UNA to ISS. The connection state should be changed to SYN-RECEIVED. Note that any other incoming control or data (combined with SYN) will be processed in the SYN-RECEIVED state, but processing of SYN and ACK should not be repeated. If the listen was not fully specified (i.e., the foreign socket was not fully specified), then the unspecified fields should be filled in now.

fourth other text or control

Any other control or text-bearing segment (not containing SYN) must have an ACK and thus would be discarded by the ACK processing. An incoming RST segment could not be valid, since it could not have been sent in response to anything sent by this incarnation of the connection. So you are unlikely to get here, but if you do, drop the segment, and return.

If the state is SYN-SENT then

first check the ACK bit

If the ACK bit is set

If SEG.ACK =< ISS, or SEG.ACK > SND.NXT, send a reset (unless the RST bit is set, if so drop the segment and return)

<SEQ=SEG.ACK><CTL=RST>

and discard the segment. Return.

If SND.UNA =< SEG.ACK =< SND.NXT then the ACK is acceptable.

second check the RST bit

If the RST bit is set

If the ACK was acceptable then signal the user "error: connection reset", drop the segment, enter CLOSED state, delete TCB, and return. Otherwise (no ACK) drop the segment and return.

third check the security and precedence

If the security/compartment in the segment does not exactly match the security/compartment in the TCB, send a reset

If there is an ACK

<SEQ=SEG.ACK><CTL=RST>

Otherwise

<SEQ=0><ACK=SEG.SEQ+SEG.LEN><CTL=RST,ACK>

If there is an ACK

The precedence in the segment must match the precedence in the TCB, if not, send a reset

<SEQ=SEG.ACK><CTL=RST>

If there is no ACK

If the precedence in the segment is higher than the precedence in the TCB then if allowed by the user and the system raise the precedence in the TCB to that in the segment, if not allowed to raise the prec then send a reset.

<SEQ=0><ACK=SEG.SEQ+SEG.LEN><CTL=RST,ACK>

If the precedence in the segment is lower than the precedence in the TCB continue.

If a reset was sent, discard the segment and return.

fourth check the SYN bit

This step should be reached only if the ACK is ok, or there is no ACK, and it the segment did not contain a RST.

If the SYN bit is on and the security/compartment and precedence are acceptable then, RCV.NXT is set to SEG.SEQ+1, IRS is set to SEG.SEQ. SND.UNA should be advanced to equal SEG.ACK (if there is an ACK), and any segments on the retransmission queue which are thereby acknowledged should be removed.

If SND.UNA > ISS (our SYN has been ACKed), change the connection state to ESTABLISHED, form an ACK segment

<SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>

and send it. Set the following variables:

SND.WND = SEG.WND
SND.WL1 = SEG.SEQ
SND.WL2 = SEG.ACK

Data or controls which were queued for transmission may be included. If there are other controls or text in the segment then continue processing at the sixth step below where the URG bit is checked, otherwise return.

Otherwise enter SYN-RECEIVED, form a SYN,ACK segment

<SEQ=ISS><ACK=RCV.NXT><CTL=SYN,ACK>

and send it. If there are other controls or text in the segment, queue them for processing after the ESTABLISHED state has been reached, return.

fifth, if neither of the SYN or RST bits is set then drop the segment and return.

Otherwise,

first check sequence number

SYN-RECEIVED STATE
ESTABLISHED STATE
FIN-WAIT-1 STATE
FIN-WAIT-2 STATE
CLOSE-WAIT STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

Segments are processed in sequence. Initial tests on arrival are used to discard old duplicates, but further processing is done in SEG.SEQ order. If a segment's contents straddle the boundary between old and new, only the new parts should be processed.

There are four cases for the acceptability test for an incoming segment:

Seg Length	Receive Window	Test
0	0	SEG.SEQ = RCV.NXT
0	>0	RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND
>0	0	not acceptable
>0	>0	RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND or RCV.NXT =< SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.WND

If the RCV.WND is zero, no segments will be acceptable, but special

allowance should be made to accept valid ACKs, URGs and RSTs.

If an incoming segment is not acceptable, an acknowledgment should be sent in reply (unless the RST bit is set, if so drop the segment and return):

<SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>

After sending the acknowledgment, drop the unacceptable segment and return.

In the following it is assumed that the segment is the idealized segment that begins at RCV.NXT and does not exceed the window. One could tailor actual segments to fit this assumption by trimming off any portions that lie outside the window (including SYN and FIN), and only processing further if the segment then begins at RCV.NXT. Segments with higher begining sequence numbers may be held for later processing.

second check the RST bit,

SYN-RECEIVED STATE

If the RST bit is set

If this connection was initiated with a passive OPEN (i.e., came from the LISTEN state), then return this connection to LISTEN state and return. The user need not be informed. If this connection was initiated with an active OPEN (i.e., came from SYN-SENT state) then the connection was refused, signal the user "connection refused". In either case, all segments on the retransmission queue should be removed. And in the active OPEN case, enter the CLOSED state and delete the TCB, and return.

ESTABLISHED
FIN-WAIT-1
FIN-WAIT-2
CLOSE-WAIT

If the RST bit is set then, any outstanding RECEIVEs and SEND should receive "reset" responses. All segment queues should be flushed. Users should also receive an unsolicited general "connection reset" signal. Enter the CLOSED state, delete the TCB, and return.

CLOSING STATE
LAST-ACK STATE
TIME-WAIT

If the RST bit is set then, enter the CLOSED state, delete the TCB, and return.

third check security and precedence

SYN-RECEIVED

If the security/compartment and precedence in the segment do not exactly match the security/compartment and precedence in the TCB then send a reset, and return.

ESTABLISHED STATE
FIN-WAIT-1
FIN-WAIT-2
CLOSE-WAIT
CLOSING

LAST-ACK
TIME-WAIT

If the security/compartment and precedence in the segment do not exactly match the security/compartment and precedence in the TCB then send a reset, any outstanding RECEIVEs and SEND should receive "reset" responses. All segment queues should be flushed. Users should also receive an unsolicited general "connection reset" signal. Enter the CLOSED state, delete the TCB, and return.

Note this check is placed following the sequence check to prevent a segment from an old connection between these ports with a different security or precedence from causing an abort of the current connection.

fourth, check the SYN bit,

SYN-RECEIVED

If the connection was initiated with a passive OPEN, then return this connection to the LISTEN state and return, otherwise

ESTABLISHED STATE
FIN-WAIT STATE-1
FIN-WAIT STATE-2
CLOSE-WAIT STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

If the SYN is in the window it is an error, send a reset, any outstanding RECEIVEs and SEND should receive "reset" responses, all segment queues should be flushed, the user should also receive an unsolicited general "connection reset" signal, enter the CLOSED state, delete the TCB, and return.

If the SYN is not in the window this step would not be reached and an ack would have been sent in the first step (sequence number check).

fifth check the ACK field,

if the ACK bit is off drop the segment and return

if the ACK bit is on

SYN-RECEIVED STATE

If SND.UNA =< SEG.ACK =< SND.NXT then enter ESTABLISHED state and continue processing. Set the following variables:

SND.WND = SEG.WND
SND.WL1 = SEG.SEQ
SND.WL2 = SEG.ACK

If the segment acknowledgment is not acceptable, form a reset segment,

<SEQ=SEG.ACK><CTL=RST>

and send it.

ESTABLISHED STATE

If SND.UNA < SEG.ACK =< SND.NXT then, set SND.UNA <- SEG.ACK. Any segments on the retransmission queue which are thereby entirely acknowledged are removed. Users should receive

positive acknowledgments for buffers which have been SENT and fully acknowledged (i.e., SEND buffer should be returned with "ok" response). If the ACK is a duplicate (SEG.ACK <= SND.UNA), it can be ignored. If the ACK acks something not yet sent (SEG.ACK > SND.NXT) then send an ACK, drop the segment, and return.

If SND.UNA <= SEG.ACK =< SND.NXT, the send window should be updated. If (SND.WL1 < SEG.SEQ or (SND.WL1 = SEG.SEQ and SND.WL2 =< SEG.ACK)), set SND.WND <- SEG.WND, set SND.WL1 <- SEG.SEQ, and set SND.WL2 <- SEG.ACK.

Note that SND.WND is an offset from SND.UNA, that SND.WL1 records the sequence number of the last segment used to update SND.WND, and that SND.WL2 records the acknowledgment number of the last segment used to update SND.WND. The check here prevents using old segments to update the window.

FIN-WAIT-1 STATE

In addition to the processing for the ESTABLISHED state, if our FIN is now acknowledged then enter FIN-WAIT-2 and continue processing in that state.

FIN-WAIT-2 STATE

In addition to the processing for the ESTABLISHED state, if the retransmission queue is empty, the user's CLOSE can be acknowledged ("ok") but do not delete the TCB.

CLOSE-WAIT STATE

Do the same processing as for the ESTABLISHED state.

CLOSING STATE

In addition to the processing for the ESTABLISHED state, if the ACK acknowledges our FIN then enter the TIME-WAIT state, otherwise ignore the segment.

LAST-ACK STATE

The only thing that can arrive in this state is an acknowledgment of our FIN. If our FIN is now acknowledged, delete the TCB, enter the CLOSED state, and return.

TIME-WAIT STATE

The only thing that can arrive in this state is a retransmission of the remote FIN. Acknowledge it, and restart the 2 MSL timeout.

sixth, check the URG bit,

ESTABLISHED STATE
FIN-WAIT-1 STATE
FIN-WAIT-2 STATE

If the URG bit is set, RCV.UP <- max(RCV.UP,SEG.UP), and signal the user that the remote side has urgent data if the urgent pointer (RCV.UP) is in advance of the data consumed. If the user has already been signaled (or is still in the "urgent mode") for this continuous sequence of urgent data, do not signal the user again.

CLOSE-WAIT STATE
CLOSING STATE

LAST-ACK STATE
TIME-WAIT

This should not occur, since a FIN has been received from the remote side. Ignore the URG.

seventh, process the segment text,

ESTABLISHED STATE
FIN-WAIT-1 STATE
FIN-WAIT-2 STATE

Once in the ESTABLISHED state, it is possible to deliver segment text to user RECEIVE buffers. Text from segments can be moved into buffers until either the buffer is full or the segment is empty. If the segment empties and carries an PUSH flag, then the user is **optionally** informed, when the buffer is returned, that a PUSH has been received.

When the TCP takes responsibility for delivering the data to the user it must also acknowledge the receipt of the data.

Once the TCP takes responsibility for the data it advances RCV.NXT over the data accepted, and adjusts RCV.WND as apporopriate to the current buffer availability. The total of RCV.NXT and RCV.WND should not be reduced.

Please note the window management suggestions.

Send an acknowledgment of the form:

<SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>

This acknowledgment should be piggybacked on a segment being transmitted if possible without incurring undue delay.

CLOSE-WAIT STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

This should not occur, since a FIN has been received from the remote side. Ignore the segment text.

eighth, check the FIN bit,

Do not process the FIN if the state is CLOSED, LISTEN or SYN-SENT since the SEG.SEQ cannot be validated; drop the segment and return.

If the FIN bit is set, signal the user "connection closing" and return any pending RECEIVEs with same message, advance RCV.NXT over the FIN, and send an acknowledgment for the FIN. Note that FIN implies PUSH for any segment text not yet delivered to the user.

SYN-RECEIVED STATE
ESTABLISHED STATE

Enter the CLOSE-WAIT state.

FIN-WAIT-1 STATE

If our FIN has been ACKed (perhaps in this segment), then enter TIME-WAIT, start the time-wait timer, turn off the other timers; otherwise enter the CLOSING state.

FIN-WAIT-2 STATE

Enter the TIME-WAIT state. Start the time-wait timer, turn off the other timers.

CLOSE-WAIT STATE

Remain in the CLOSE-WAIT state.

CLOSING STATE

Remain in the CLOSING state.

LAST-ACK STATE

Remain in the LAST-ACK state.

TIME-WAIT STATE

Remain in the TIME-WAIT state. Restart the 2 MSL time-wait timeout.

and return.

USER TIMEOUT

For any state if the user timeout expires, flush all queues, signal the user "error: connection aborted due to user timeout" in general and for any outstanding calls, delete the TCB, enter the CLOSED state and return.

Discussion

It would be better to notify the application of the timeout rather than letting TCP force the connection closed, see TCP Conneciton Failures.

RETRANSMISSION TIMEOUT

For any state if the retransmission timeout expires on a segment in the retransmission queue, send the segment at the front of the retransmission queue again, reinitialize the retransmission timer, and return.

TIME-WAIT TIMEOUT

If the time-wait timeout expires on a connection delete the TCB, enter the CLOSED state and return.

GLOSSARY

1822

BBN Report 1822, "The Specification of the Interconnection of a Host and an IMP". The specification of interface between a host and the ARPANET.

ACK

A control bit (acknowledge) occupying no sequence space, which indicates that the acknowledgment field of this segment specifies the next sequence number the sender of this segment is expecting to receive, hence acknowledging receipt of all previous sequence numbers.

ARPANET message

The unit of transmission between a host and an IMP in the ARPANET. The maximum size is about 1012 octets (8096 bits).

ARPANET packet

A unit of transmission used internally in the ARPANET between IMPs. The maximum size is about 126 octets (1008 bits).

connection

A logical communication path identified by a pair of sockets.

datagram

A message sent in a packet switched computer communications network.

Destination Address

The destination address, usually the network and host identifiers.

FIN

A control bit (finis) occupying one sequence number, which indicates that the sender will send no more data or control occupying sequence space.

fragment

A portion of a logical unit of data, in particular an internet fragment is a portion of an internet datagram.

FTP

A file transfer protocol.

header

Control information at the beginning of a message, segment, fragment, packet or block of data.

host

A computer. In particular a source or destination of messages from the point of view of the communication network.

Identification

An Internet Protocol field. This identifying value assigned by the sender aids in assembling the fragments of a datagram.

IMP

The Interface Message Processor, the packet switch of the ARPANET.

internet address

A source or destination address specific to the host level.

internet datagram

The unit of data exchanged between an internet module and the higher level protocol together with the internet header.

internet fragment

A portion of the data of an internet datagram with an internet header.

IP

Internet Protocol.

IRS

The Initial Receive Sequence number. The first sequence number used by the sender on a connection.

ISN

The Initial Sequence Number. The first sequence number used on a connection, (either ISS or IRS). Selected on a clock based procedure.

ISS

The Initial Send Sequence number. The first sequence number used by the sender on a connection. leader Control information at the beginning of a message or block of data. In particular, in the ARPANET, the control information on an ARPANET message at the host-IMP interface.

left sequence

This is the next sequence number to be acknowledged by the data receiving TCP (or the lowest currently unacknowledged sequence number) and is sometimes referred to as the left edge of the send window.

local packet

The unit of transmission within a local network. module An implementation, usually in software, of a protocol or other procedure.

MSL

Maximum Segment Lifetime, the time a TCP segment can exist in the internetwork system. Arbitrarily defined to be 2 minutes.

octet

An eight bit byte.

Options

An Option field may contain several options, and each option may be several octets in length. The options are used primarily in testing situations; for example, to carry timestamps. Both the Internet Protocol and TCP provide for options fields.

packet

A package of data with a header which may or may not be logically complete. More often a physical packaging than a logical packaging of data.

port

The portion of a socket that specifies which logical input or output channel of a process is associated with the data.

process

A program in execution. A source or destination of data from the point of view of the TCP or other host-to-host protocol.

PUSH

A control bit occupying no sequence space, indicating that this segment contains data that must be pushed through to the receiving user.

RCV.NXT receive next sequence number

RCV.UP

receive urgent pointer

RCV.WND

receive window receive next sequence number This is the next sequence number the local TCP is expecting to receive.

receive window

This represents the sequence numbers the local (receiving) TCP is willing to receive. Thus, the local TCP considers that segments overlapping the range RCV.NXT to RCV.NXT + RCV.WND - 1 carry acceptable data or control. Segments containing sequence numbers entirely outside of this range are considered duplicates and discarded.

RST

A control bit (reset), occupying no sequence space, indicating that the receiver should delete the connection without further interaction. The receiver can determine, based on the sequence number and acknowledgment fields of the incoming segment, whether it should honor the reset command or ignore it. In no case does receipt of a segment containing RST give rise to a RST in response.

RTP

Real Time Protocol: A host-to-host protocol for communication of time critical information.

SEG.ACK

segment acknowledgment

SEG.LEN

segment length

SEG.PRC

segment precedence value

SEG.SEQ

segment sequence

SEG.UP

segment urgent pointer field

SEG.WND

segment window field segment A logical unit of data, in particular a TCP segment is the unit of data transfered between a pair of TCP modules.

segment acknowledgment

The sequence number in the acknowledgment field of the arriving segment.

segment length

The amount of sequence number space occupied by a segment, including any controls which occupy sequence space.

segment sequence

The number in the sequence field of the arriving segment. send sequence This is the next sequence number the local (sending) TCP will use on the connection. It is initially selected from an initial sequence number curve (ISN) and is incremented for each octet of data or sequenced control transmitted.

send window

This represents the sequence numbers which the remote (receiving) TCP is willing to receive. It is the value of the window field specified in segments from the remote (data receiving) TCP. The range of new sequence numbers which may be emitted by a TCP lies between SND.NXT and SND.UNA + SND.WND - 1. (Retransmissions of sequence numbers between SND.UNA and SND.NXT are expected, of course.)

SND.NXT

send sequence

SND.UNA

left sequence

SND.UP

send urgent pointer

SND.WL1

segment sequence number at last window update

SND.WL2

segment acknowledgment number at last window update

SND.WND

send window

socket

An address which specifically includes a port identifier, that is, the concatenation of an Internet Address with a TCP port.

Source Address

The source address, usually the network and host identifiers.

SYN

A control bit in the incoming segment, occupying one sequence number, used at the initiation of a connection, to indicate where the sequence numbering will start.

TCB

Transmission control block, the data structure that records the state of a connection.

TCB.PRC

The precedence of the connection.

TCP

Transmission Control Protocol: A host-to-host protocol for reliable communication in internetwork environments.

TOS

Type of Service, an Internet Protocol field.

Type of Service

An Internet Protocol field which indicates the type of service for this internet fragment.

URG

A control bit (urgent), occupying no sequence space, used to indicate that the receiving user should be notified to do urgent processing as long as there is data to be consumed with sequence numbers less than the value indicated in the urgent pointer.

urgent pointer

A control field meaningful only when the URG bit is on. This field communicates the value of the urgent pointer which indicates the data octet associated with the sending user's urgent call.

REFERENCES

[1] Cerf, V., and R. Kahn, "A Protocol for Packet Network Intercommunication", IEEE Transactions on Communications, Vol. COM-22, No. 5, pp 637-648, May 1974.

[2] Postel, J. (ed.), "Internet Protocol - DARPA Internet Program Protocol Specification", RFC 791, USC/Information Sciences Institute, September 1981.

[3] Dalal, Y. and C. Sunshine, "Connection Management in Transport Protocols", Computer Networks, Vol. 2, No. 6, pp. 454-473, December 1978.

[4] Postel, J., "Assigned Numbers", RFC 790, USC/Information Sciences Institute, September 1981.

RFC-795 Service Mappings

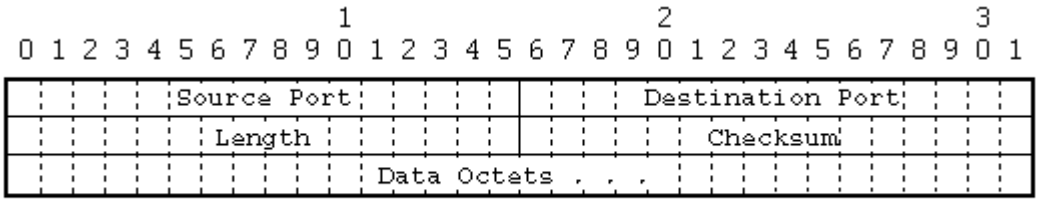
J. Postel
USC/Information Sciences Institute
September 1981

This memo is no longer valid but is included here for historical purposes.

This memo describes the relationship between the Internet Protocol (IP) Type of Service and the service parameters of specific networks.

The IP Type of Service has the following fields:

Bits 0-2: Precedence.
Bit 3: 0 = Normal Delay, 1 = Low Delay.
Bit 4: 0 = Normal Throughput, 1 = High Throughput.
Bit 5: 0 = Normal Relibility, 1 = High Relibility.
Bit 6-7: Reserved for Future Use.



User Datagram Header Format

111 - Network Control
110 - Internetwork Control
101 - CRITIC/ECP
100 - Flash Override
011 - Flash
010 - Immediate
001 - Priority
000 - Routine

The individual networks listed here have very different and specific service choices.

AUTODIN II
ARPANET
PRNET
SATNET

RFC-795 Service Mappings

AUTODIN II

The service choices are in two parts: Traffic Acceptance Catagories, and Application Type. The Traffic Acceptance Catagories can be mapped into and out of the IP TOS precedence reasonably directly. The Application types can be mapped into the remaining IP TOS fields as follows.

TA	DELAY	THROUGHPUT	RELIABILITY
I/A	1	0	0
Q/R	0	0	0
B1	0	1	0
B2	0	1	1

DTR	TA
000	Q/R
001	Q/R
010	B1
011	B2
100	I/A
101	I/A
110	I/A
111	error

RFC-795 Service Mappings

ARPANET

The service choices are in quite limited. There is one priority bit that can be mapped to the high order bit of the IP TOS precedence. The other choices are to use the regular ("Type 0") messages vs. the uncontrolled ("Type 3") messages, or to use single packet vs. multipacket messages. The mapping of ARPANET parameters into IP TOS parameters can be as follows.

Type	Size	DELAY	THROUGHPUT	RELIABILITY
0	S	1	0	0
0	M	0	0	0
3	S	1	0	0
3	M	not allowed		

DTR	Type	Size
000	0	M
001	0	M
010	0	M
011	0	M
100	3	S
101	0	S
110	3	S
111	error	

RFC-795 Service Mappings

PRNET

There is no priority indication. The two choices are to use the station routing vs. point-to-point routing, or to require acknowledgments vs. having no acknowledgments. The mapping of PRNET parameters into IP TOS parameters can be as follows.

Routing	Acks	DELAY	THROUGHPUT	RELIABILITY
ptp	no	1	0	0
ptp	yes	1	0	1
station	no	0	0	0
station	yes	0	0	1

DTR	Routing	Acks
000	station	no
001	station	yes
010	station	no
011	station	yes
100	ptp	no
101	ptp	yes
110	ptp	no
111	ptp	yes

SATNET

There is no priority indication. The four choices are to use the block vs. stream type, to select one of four delay catagories, to select one of two holding time strategies, or to request one of three reliability levels. The mapping of SATNET parameters into IP TOS parameters can thus quite complex there being 2*4*2*3=48 distinct possibilities.

RFC-796 Address Mappings

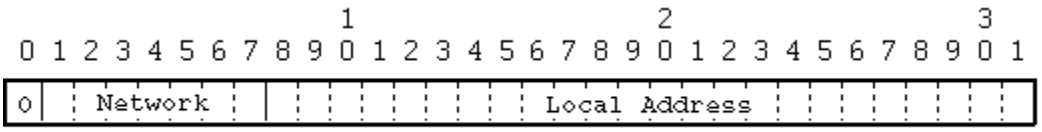
J. Postel
USC/Information Sciences Institute
September 1981

Internet Addresses

This memo describes the relationship between address fields used in the Internet Protocol and several specific networks.

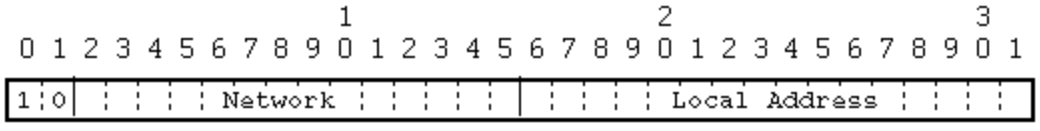
An internet address is a 32 bit quantity, with several codings as shown below.

The first type (or class a) of address has a 7-bit network number and a 24-bit local address.



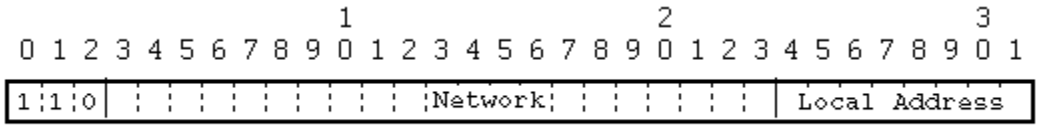
Class A Address

The second type (or class b) of address has a 14-bit network number and a 16-bit local address.



Class B Address

The third type (or class c) of address has a 21-bit network number and a 8-bit local address.



Class C Address

The local address carries information to address a host in the network identified by the network number. Since each network has a particular address format and length, the following section describes the mapping between internet local addresses and the actual address format used in the particular network.

Internet to Local Net Address Mappings

The following transformations are used to convert internet addresses to local net addresses and vice versa:

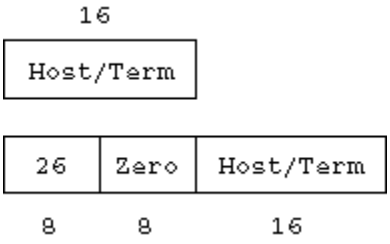
AUTODIN II
ARPANET
DCNs
EDN
LCSNET
PRNET
SATNET
WBCNET

RFC-796 Address Mappings - Internet to Local Net Address Mappings

AUTODIN II

The AUTODIN II has 16 bit subscriber addresses which identify either a host or a terminal. These addresses may be assigned independent of location. The 16 bit AUTODIN II address is located in the 24 bit internet local address as shown below.

The network number of the AUTODIN II is 26 (Class A).



RFC-796 Address Mappings - Internet to Local Net Address Mappings

ARPANET

The ARPANET (with 96 bit leaders) has 24 bit addresses. The 24 bits are assigned to host, logical host, and IMP leader fields as illustrated below. These 24 bit addresses are used directly for the 24 bit local address of the internet address. However, the ARPANET IMPs do not yet support this form of logical addressing so the logical host field is set to zero in the leader.

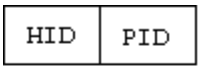
The network number of the ARPANET is 10 (Class A).

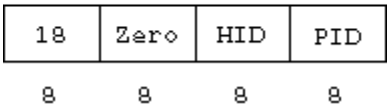
RFC-796 Address Mappings - Internet to Local Net Address Mappings

DCNs

The Distributed Computing Networks (DCNs) at COMSAT and UCL use 16 bit addresses divided into an 8 bit host identifier (HID), and an 8 bit process identifier (PID). The format locates these 16 bits in the low order 16 bits of the 24 bit internet address, as shown below.

The network number of the COMSAT-DCN is 29 (Class A), and of the UCL-DCN is 30 (Class A).





EDN

The Experimental Data Network at the Defense Communication Engineering Center (DCEC) uses the same type of addresses as the ARPANET (with 96 bit leaders) and has 24 bit addresses. The 24 bits are assigned to host, logical host, and IMP leader fields as illustrated below. These 24 bit addresses are used directly for the 24 bit local address of the internet address. However, the IMPs do not yet support this form of logical addressing so the logical host field is set to zero in the leader.

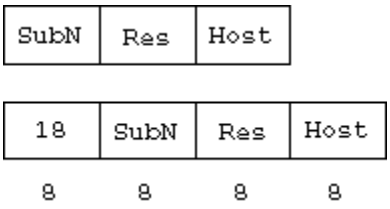
The network number of the EDN is 21 (Class A).

RFC-796 Address Mappings - Internet to Local Net Address Mappings

LCSNET

The LCS NET at MIT's Laboratory for Computer Science uses 32 bit addresses of several formats. The most common format locates the low order 24 bits of the 32 bit LCS NET address in the 24 bit internet local address, as shown below.

The network number of the LCS NET is 18 (Class A).



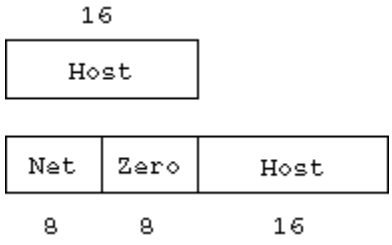
RFC-796 Address Mappings - Internet to Local Net Address Mappings

PRNET

The Packet Radio networks use 16 bit addresses. These are independent of location (indeed the hosts may be mobile). The 16 bit PRNET addresses are located in the 24 bit internet local address as shown below.

The network numbers of the PRNETs are:

BBN-PR	1 (Class A)
SF-PR-1	2 (Class A)
SILL-PR	5 (Class A)
SF-PR-2	6 (Class A)
BRAGG-PR	9 (Class A)
DC-PR	20 (Class A)

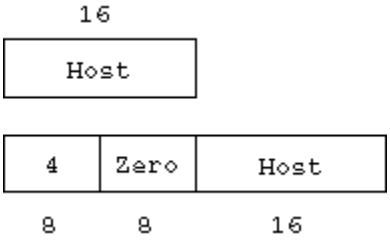


RFC-796 Address Mappings - Internet to Local Net Address Mappings

SATNET

The Atlantic Satellite Packet Network has 16 bit addresses for hosts. These addresses may be assigned independent of location (i.e., ground station). It is also possible to assign several addresses to one physical host, so the addresses are logical addresses. The 16 bit SATNET address is located in the 24 bit internet local address as shown below.

The network number of the SATNET is 4 (Class A).

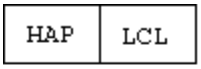


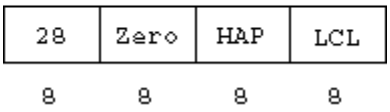
RFC-796 Address Mappings - Internet to Local Net Address Mappings

WBCNET

The Wideband Communication Satellite Packet Network (WBCNET) Host Access Protocol (HAP) has 16 bit addresses for hosts. It is possible to assign several addresses to one physical host, so the addresses are logical addresses. The 16 bit WBCNET address is divided into a HAP Number field and a Local Address field, and is located in the 24 bit internet local address as shown below.

The network number of the WBCNET is 28 (Class A).





RFC-813 Window and Acknowledgement Strategy in TCP

David D. Clark
MIT Laboratory for Computer Science
July 1982

This document describes implementation strategies to deal with two mechanisms in TCP, the window and the acknowledgement. These mechanisms are described in the specification document, but it is possible, while complying with the specification, to produce implementations which yield very bad performance. Happily, the pitfalls possible in window and acknowledgement strategies are very easy to avoid.

It is a much more difficult exercise to verify the performance of a specification than the correctness. Certainly, we have less experience in this area, and we certainly lack any useful formal technique. Nonetheless, it is important to attempt a specification in this area, because different implementors might otherwise choose superficially reasonable algorithms which interact poorly with each other. This document presents a particular set of algorithms which have received testing in the field, and which appear to work properly with each other. With more experience, these algorithms may become part of the formal specification: until such time their use is recommended.

RFC-813 Window and Acknowledgement Strategy in TCP

The Mechanisms

The acknowledgement mechanism is at the heart of TCP. Very simply, when data arrives at the recipient, the protocol requires that it send back an acknowledgement of this data. The protocol specifies that the bytes of data are sequentially numbered, so that the recipient can acknowledge data by naming the highest numbered byte of data it has received, which also acknowledges the previous bytes (actually, it identifies the first byte of data which it has not yet received, but this is a small detail). The protocol contains only a general assertion that data should be acknowledged promptly, but gives no more specific indication as to how quickly an acknowledgement must be sent, or how much data should be acknowledged in each separate acknowledgement.

The window mechanism is a flow control tool. Whenever appropriate, the recipient of data returns to the sender a number, which is (more or less) the size of the buffer which the receiver currently has available for additional data. This number of bytes, called the window, is the maximum which the sender is permitted to transmit until the receiver returns some additional window. Sometimes, the receiver will have no buffer space available, and will return a window value of zero. Under these circumstances,the protocol requires the sender to send a small segment to the receiver now and then, to see if more data is accepted. If the window remains closed at zero for some substantial period, and the sender can obtain no response from the receiver, the protocol requires the sender to conclude that the receiver has failed, and to close the connection. Again, there is very little performance information in the specification, describing under what circumstances the window should be increased, and how the sender should respond to such revised information.

A bad implementation of the window algorithm can lead to extremely poor performance overall. The degradations which occur in throughput and CPU utilizations can easily be several factors of ten, not just a fractional increase. This particular phenomenon is specific enough that it has been given the name of Silly Window Syndrome, or SWS. Happily SWS is easy to avoid if a few simple rules are observed. The most important function of this memo is to describe SWS, so that implementors will understand the general nature of the problem, and to describe algorithms which will prevent its occurrence. This document also describes performance enhancing algorithms which relate to acknowledgement, and discusses the way acknowledgement and window algorithms interact as part of SWS.

RFC-813 Window and Acknowledgement Strategy in TCP

Silly Window Syndrome

In order to understand SWS, we must first define two new terms. Superficially, the window mechanism is very simple: there is a number, called "the window", which is returned from the receiver to the sender. However, we must have a more detailed way of talking about the meaning of this number. The receiver of data computes a value which we will call the "offered window". In a simple case, the offered window corresponds to the amount of buffer space available in the receiver. This correspondence is not necessarily exact, but is a suitable model for the discussion to follow. It is the offered window which is actually transmitted back from the receiver to the sender. The sender uses the offered window to compute a different value, the "usable window", which is the offered window minus the amount of outstanding unacknowledged data. The usable window is less than or equal to the offered window, and can be much smaller.

Consider the following simple example. The receiver initially provides an offered window of 1,000. The sender uses up this window by sending five segments of 200 bytes each. The receiver, on processing the first of these segments, returns an acknowledgement which also contains an updated window value. Let us assume that the receiver of the data has removed the first 200 bytes from the buffer, so that the receiver once again has 1,000 bytes of available buffer. Therefore, the receiver would return, as before, an offered window of 1,000 bytes. The sender, on receipt of this first acknowledgement, now computes the additional number of bytes which may be sent. In fact, of the 1,000 bytes which the recipient is prepared to receive at this time, 800 are already in transit, having been sent in response to the previous offered window. In this case, the usable window is only 200 bytes.

Let us now consider how SWS arises. To continue the previous example, assume that at some point, when the sender computes a useable window of 200 bytes, it has only 50 bytes to send until it reaches a "push" point. It thus sends 50 bytes in one segment, and 150 bytes in the next segment. Sometime later, this 50-byte segment will arrive at the recipient, which will process and remove the 50 bytes and once again return an offered window of 1,000 bytes. However, the sender will now compute that there are 950 bytes in transit in the network, so that the useable window is now only 50 bytes. Thus, the sender will once again send a 50 byte segment, even though there is no longer a natural boundary to force it.

In fact, whenever the acknowledgement of a small segment comes back, the useable window associated with that acknowledgement will cause another segment of the same small size to be sent, until some abnormality breaks the pattern. It is easy to see how small segments arise, because natural boundaries in the data occasionally cause the sender to take a computed useable window and divide it up between two segments. Once that division has occurred, there is no natural way for those useable window allocations to be recombined; thus the breaking up of the useable window into small pieces will persist.

Thus, SWS is a degeneration in the throughput which develops over time, during a long data transfer. If the sender ever stops, as for example when it runs out of data to send, the receiver will eventually acknowledge all the outstanding data, so that the useable window computed by the sender will equal the full offered window of the receiver. At this point the situation will have healed, and further data transmission over the link will occur efficiently. However, in large file transfers, which occur without interruption, SWS can cause appalling performance. The network between the sender and the receiver becomes clogged with many small segments, and an equal number of acknowledgements, which in turn causes lost segments, which triggers massive retransmission. Bad cases of SWS have been seen in which the average segment size was one-tenth of the size the sender and receiver were prepared to deal with, and the average number of retransmission per successful segments

sent was five.

Happily, SWS is trivial to avoid. The following sections describe two algorithms, one executed by the sender, and one by the receiver, which appear to eliminate SWS completely. Actually, either algorithm by itself is sufficient to prevent SWS, and thus protect a host from a foreign implementation which has failed to deal properly with this problem. The two algorithms taken together produce an additional reduction in CPU consumption, observed in practice to be as high as a factor of four.

RFC-813 Window and Acknowledgement Strategy in TCP

Improved Window Algorithms

The receiver of data can take a very simple step to eliminate SWS. When it disposes of a small amount of data, it can artificially reduce the offered window in subsequent acknowledgements, so that the useable window computed by the sender does not permit the sending of any further data. At some later time, when the receiver has processed a substantially larger amount of incoming data, the artificial limitation on the offered window can be removed all at once, so that the sender computes a sudden large jump rather than a sequence of small jumps in the useable window.

At this level, the algorithm is quite simple, but in order to determine exactly when the window should be opened up again, it is necessary to look at some of the other details of the implementation. Depending on whether the window is held artificially closed for a short or long time, two problems will develop. The one we have already discussed -- never closing the window artificially -- will lead to SWS. On the other hand, if the window is only opened infrequently, the pipeline of data in the network between the sender and the receiver may have emptied out while the sender was being held off, so that a delay is introduced before additional data arrives from the sender. This delay does reduce throughput, but it does not consume network resources or CPU resources in the process, as does SWS. Thus, it is in this direction that one ought to overcompensate. For a simple implementation, a rule of thumb that seems to work in practice is to artificially reduce the offered window until the reduction constitutes one half of the available space, at which point increase the window to advertise the entire space again. In any event, one ought to make the chunk by which the window is opened at least permit one reasonably large segment. (If the receiver is so short of buffers that it can never advertise a large enough buffer to permit at least one large segment, it is hopeless to expect any sort of high throughput.)

Note: The algorithm described below for the sender is **obsolete** and **should not** be used. An improved algorithm is given in the section "When to Send Data**".**

There is an algorithm that the sender can use to achieve the same effect described above: a very simple and elegant rule first described by Michael Greenwald at MIT. The sender of the data uses the offered window to compute a useable window, and then compares the useable window to the offered window, and refrains from sending anything if the ratio of useable to offered is less than a certain fraction. Clearly, if the computed useable window is small compared to the offered window, this means that a substantial amount of previously sent information is still in the pipeline from the sender to the receiver, which in turn means that the sender can count on being granted a larger useable window in the future. Until the useable window reaches a certain amount, the sender should simply refuse to send anything.

Simple experiments suggest that the exact value of the ratio is not very important, but that a value of about 25 percent is sufficient to avoid SWS and achieve reasonable throughput, even for machines with a small offered window. An additional enhancement which might help throughput would be to attempt to hold off sending until one can send a maximum size segment. Another enhancement would be to send anyway, even if the ratio is small, if the useable window is sufficient to hold the data available up to the next "push point".

This algorithm at the sender end is very simple. Notice that it is not necessary to set a timer to protect against protocol lockup when postponing the send operation. Further acknowledgements, as they arrive, will inevitably change the ratio of offered to useable window. (To see this, note that when all the data in the catanet pipeline has arrived at the receiver, the resulting acknowledgement must yield an offered window and useable window that equal each other.) If the expected acknowledgements do not arrive, the retransmission

mechanism will come into play to assure that something finally happens. Thus, to add this algorithm to an existing TCP implementation usually requires one line of code. As part of the send algorithm it is already necessary to compute the useable window from the offered window. It is a simple matter to add a line of code which, if the ratio is less than a certain percent, sets the useable window to zero. The results of SWS are so devastating that no sender should be without this simple piece of insurance.

Improved Acknowledgement Algorithms

In the beginning of this paper, an overly simplistic implementation of TCP was described, which led to SWS. One of the characteristics of this implementation was that the recipient of data sent a separate acknowledgement for every segment that it received. This compulsive acknowledgement was one of the causes of SWS, because each acknowledgement provided some new useable window, but even if one of the algorithms described above is used to eliminate SWS, overly frequent acknowledgement still has a substantial problem, which is that it greatly increases the processing time at the sender's end. Measurement of TCP implementations, especially on large operating systems, indicate that most of the overhead of dealing with a segment is not in the processing at the TCP or IP level, but simply in the scheduling of the handler which is required to deal with the segment. A steady dribble of acknowledgements causes a high overhead in scheduling, with very little to show for it. This waste is to be avoided if possible.

There are two reasons for prompt acknowledgement. One is to prevent retransmission. We will discuss later how to determine whether unnecessary retransmission is occurring. The other reason one acknowledges promptly is to permit further data to be sent. However, the previous section makes quite clear that it is not always desirable to send a little bit of data, even though the receiver may have room for it. Therefore, one can state a general rule that under normal operation, the receiver of data need not, and for efficiency reasons should not, acknowledge the data unless either the acknowledgement is intended to produce an increased useable window, is necessary in order to prevent retransmission or is being sent as part of a reverse direction segment being sent for some other reason. We will consider an algorithm to achieve these goals.

Only the recipient of the data can control the generation of acknowledgements. Once an acknowledgement has been sent from the receiver back to the sender, the sender must process it. Although the extra overhead is incurred at the sender's end, it is entirely under the receiver's control. Therefore, we must now describe an algorithm which occurs at the receiver's end. Obviously, the algorithm must have the following general form; sometimes the receiver of data, upon processing a segment, decides not to send an acknowledgement now, but to postpone the acknowledgement until some time in the future, perhaps by setting a timer. The peril of this approach is that on many large operating systems it is extremely costly to respond to a timer event, almost as costly as to respond to an incoming segment. Clearly, if the receiver of the data, in order to avoid extra overhead at the sender end, spends a great deal of time responding to timer interrupts, no overall benefit has been achieved, for efficiency at the sender end is achieved by great thrashing at the receiver end. We must find an algorithm that avoids both of these perils.

The following scheme seems a good compromise. The receiver of data will refrain from sending an acknowledgement under certain circumstances, in which case it must set a timer which will cause the acknowledgement to be sent later. However, the receiver should do this only where it is a reasonable guess that some other event will intervene and prevent the necessity of the timer interrupt. The most obvious event on which to depend is the arrival of another segment. So, if a segment arrives, postpone sending an acknowledgement if both of the following conditions hold. First, the push bit is not set in the segment, since it is a reasonable assumption that there is more data coming in a subsequent segment. Second, there is no revised window information to be sent back.

This algorithm will insure that the timer, although set, is seldom used. The interval of the timer is related to the expected inter- segment delay, which is in turn a function of the particular network through which the data is flowing. For the Arpanet, a reasonable interval seems to be 200 to 300 milliseconds. Appendix A describes an adaptive algorithm for

measuring this delay.

The section on improved window algorithms described both a receiver algorithm and a sender algorithm, and suggested that both should be used. The reason for this is now clear. While the sender algorithm is extremely simple, and useful as insurance, the receiver algorithm is required in order that this improved acknowledgement strategy work. If the receipt of every segment causes a new window value to be returned, then of necessity an acknowledgement will be sent for every data segment. When, according to the strategy of the previous section, the receiver determines to artificially reduce the offered window, that is precisely the circumstance under which an acknowledgement need not be sent. When the receiver window algorithm and the receiver acknowledgement algorithm are used together, it will be seen that sending an acknowledgement will be triggered by one of the following events. First, a push bit has been received. Second, a temporary pause in the data stream is detected. Third, the offered window has been artificially reduced to one-half its actual value.

In the beginning of this section, it was pointed out that there are two reasons why one must acknowledge data. Our consideration at this point has been concerned only with the first, that an acknowledgement must be returned as part of triggering the sending of new data. It is also necessary to acknowledge whenever the failure to do so would trigger retransmission by the sender. Since the retransmission interval is selected by the sender, the receiver of the data cannot make a precise determination of when the acknowledgement must be sent. However, there is a rough rule the sender can use to avoid retransmission, provided that the receiver is reasonably well behaved.

We will assume that sender of the data uses the optional algorithm described in the TCP specification, in which the roundtrip delay is measured using an exponential decay smoothing algorithm. Retransmission of a segment occurs if the measured delay for that segment exceeds the smoothed average by some factor. To see how retransmission might be triggered, one must consider the pattern of segment arrivals at the receiver. The goal of our strategy was that the sender should send off a number of segments in close sequence, and receive one acknowledgement for the whole burst. The acknowledgement will be generated by the receiver at the time that the last segment in the burst arrives at the receiver. (To ensure the prompt return of the acknowledgement, the sender could turn on the "push" bit in the last segment of the burst.) The delay observed at the sender between the initial transmission of a segment and the receipt of the acknowledgement will include both the network transit time, plus the holding time at the receiver. The holding time will be greatest for the first segments in the burst, and smallest for the last segments in the burst. Thus, the smoothing algorithm will measure a delay which is roughly proportional to the average roundtrip delay for all the segments in the burst. Problems will arise if the average delay is substantially smaller than the maximum delay and the smoothing algorithm used has a very small threshold for triggering retransmission. The widest variation between average and maximum delay will occur when network transit time is negligible, and all delay is processing time. In this case, the maximum will be twice the average (by simple algebra) so the threshold that controls retransmission should be somewhat more than a factor of two.

In practice, retransmission of the first segments of a burst has not been a problem because the delay measured consists of the network roundtrip delay, as well as the delay due to withholding the acknowledgement, and the roundtrip tends to dominate except in very low roundtrip time situations (such as when sending to one's self for test purposes). This low roundtrip situation can be covered very simply by including a minimum value below which the roundtrip estimate is not permitted to drop.

In our experiments with this algorithm, retransmission due to faulty calculation of the roundtrip delay occurred only once, when the parameters of the exponential smoothing algorithm had been misadjusted so that they were only taking into account the last two or three segments sent. Clearly, this will cause trouble since the last two or three segments of

any burst are the ones whose holding time at the receiver is minimal, so the resulting total estimate was much lower than appropriate. Once the parameters of the algorithm had been adjusted so that the number of segments taken into account was approximately twice the number of segments in a burst of average size, with a threshold factor of 1. 5, no further retransmission has ever been identified due to this problem, including when sending to ourself and when sending over high delay nets.

RFC-813 Window and Acknowledgement Strategy in TCP

Conservative Vs. Optimistic Windows

According to the TCP specification, the offered window is presumed to have some relationship to the amount of data which the receiver is actually prepared to receive. However, it is not necessarily an exact correspondence. We will use the term "conservative window" to describe the case where the offered window is precisely no larger than the actual buffering available. The drawback to conservative window algorithms is that they can produce very low throughput in long delay situations. It is easy to see that the maximum input of a conservative window algorithm is one bufferfull every roundtrip delay in the net, since the next bufferfull cannot be launched until the updated window/acknowledgement information from the previous transmission has made the roundtrip.

In certain cases, it may be possible to increase the overall throughput of the transmission by increasing the offered window over the actual buffer available at the receiver. Such a strategy we will call an "optimistic window" strategy. The optimistic strategy works if the network delivers the data to the recipient sufficiently slowly that it can process the data fast enough to keep the buffer from overflowing. If the receiver is faster than the sender, one could, with luck, permit an infinitely optimistic window, in which the sender is simply permitted to send full-speed. If the sender is faster than the receiver, however, and the window is too optimistic, then some segments will cause a buffer overflow, and will be discarded. Therefore, the correct strategy to implement an optimistic window is to increase the window size until segments start to be lost. This only works if it is possible to detect that the segment has been lost. In some cases, it is easy to do, because the segment is partially processed inside the receiving host before it is thrown away. In other cases, overflows may actually cause the network interface to be clogged, which will cause the segments to be lost elsewhere in the net. It is inadvisable to attempt an optimistic window strategy unless one is certain that the algorithm can detect the resulting lost segments. However, the increase in throughput which is possible from optimistic windows is quite substantial. Any systems with small buffer space should seriously consider the merit of optimistic windows.

The selection of an appropriate window algorithm is actually more complicated than even the above discussion suggests. The following considerations are not presented with the intention that they be incorporated in current implementations of TCP, but as background for the sophisticated designer who is attempting to understand how his TCP will respond to a variety of networks, with different speed and delay characteristics. The particular pattern of windows and acknowledgements sent from receiver to sender influences two characteristics of the data being sent. First, they control the average data rate. Clearly, the average rate of the sender cannot exceed the average rate of the receiver, or long-term buffer overflow will occur. Second, they influence the burstiness of the data coming from the sender. Burstiness has both advantages and disadvantages. The advantage of burstiness is that it reduces the CPU processing necessary to send the data. This follows from the observed fact, especially on large machines, that most of the cost of sending a segment is not the TCP or IP processing, but the scheduling overhead of getting started.

On the other hand, the disadvantage of burstiness is that it may cause buffers to overflow, either in the eventual recipient, which was discussed above, or in an intermediate gateway, a problem ignored in this paper. The algorithms described above attempts to strike a balance between excessive burstiness, which in the extreme cases can cause delays because a burst is not requested soon enough, and excessive fragmentation of the data stream into small segments, which we identified as Silly Window Syndrome.

Under conditions of extreme delay in the network, none of the algorithms described above will achieve adequate throughput. Conservative window algorithms have a predictable

throughput limit, which is one windowfull per roundtrip delay. Attempts to solve this by optimistic window strategies may cause buffer overflows due to the bursty nature of the arriving data. A very sophisticated way to solve this is for the receiver, having measured by some means the roundtrip delay and intersegment arrival rate of the actual connection, to open his window, not in one optimistic increment of gigantic proportion, but in a number of smaller optimistic increments, which have been carefully spaced using a timer so that the resulting smaller bursts which arrive are each sufficiently small to fit into the existing buffers. One could visualize this as a number of requests flowing backwards through the net which trigger in return a number of bursts which flow back spaced evenly from the sender to the receiver. The overall result is that the receiver uses the window mechanism to control the burstiness of the arrivals, and the average rate.

To my knowledge, no such strategy has been implemented in any TCP. First, we do not normally have delays high enough to require this kind of treatment. Second, the strategy described above is probably not stable unless it is very carefully balanced. Just as buses on a single bus route tend to bunch up, bursts which start out equally spaced could well end up piling into each other, and forming the single large burst which the receiver was hoping to avoid. It is important to understand this extreme case, however, in order to understand the limits beyond which TCP, as normally implemented, with either conservative or simple optimistic windows can be expected to deliver throughput which is a reasonable percentage of the actual network capacity.

Conclusions

This paper describes three simple algorithms for performance enhancement in TCP, one at the sender end and two at the receiver. The sender algorithm is to refrain from sending if the useable window is smaller than 25 percent of the offered window. The receiver algorithms are first, to artificially reduce the offered window when processing new data if the resulting reduction does not represent more than some fraction, say 50 percent, of the actual space available, and second, to refrain from sending an acknowledgment at all if two simple conditions hold.

Either of these algorithms will prevent the worst aspects of Silly Window Syndrome, and when these algorithms are used together, they will produce substantial improvement in CPU utilization, by eliminating the process of excess acknowledgements.

Preliminary experiments with these algorithms suggest that they work, and work very well. Both the sender and receiver algorithms have been shown to eliminate SWS, even when talking to fairly silly algorithms at the other end. The Multics mailer, in particular, had suffered substantial attacks of SWS while sending large mail to a number of hosts. We believe that implementation of the sender side algorithm has eliminated every known case of SWS detected in our mailer. Implementation of the receiver side algorithm produced substantial improvements of CPU time when Multics was the sending system. Multics is a typical large operating system, with scheduling costs which are large compared to the actual processing time for protocol handlers. Tests were done sending from Multics to a host which implemented the SWS suppression algorithm, and which could either refrain or not from sending acknowledgements on each segment. As predicted, suppressing the return acknowledgements did not influence the throughput for large data transfer at all, since the throttling effect was elsewhere. However, the CPU time required to process the data at the Multics end was cut by a factor of four (In this experiment, the bursts of data which were being sent were approximately eight segments. Thus, the number of acknowledgements in the two experiments differed by a factor of eight.)

An important consideration in evaluating these algorithms is that they must not cause the protocol implementations to deadlock. All of the recommendations in this document have the characteristic that they suggest one refrain from doing something even though the protocol specification permits one to do it. The possibility exists that if one refrains from doing something now one may never get to do it later, and both ends will halt, even though it would appear superficially that the transaction can continue.

Formally, the idea that things continue to work is referred to as "liveness". One of the defects of ad hoc solutions to performance problems is the possibility that two different approaches will interact to prevent liveness. It is believed that the algorithms described in this paper are always live, and that is one of the reasons why there is a strong advantage in uniform use of this particular proposal, except in cases where it is explicitly demonstrated not to work.

The argument for liveness in these solutions proceeds as follows. First, the sender algorithm can only be stopped by one thing, a refusal of the receiver to acknowledge sent data. As long as the receiver continues to acknowledge data, the ratio of useable window to offered window will approach one, and eventually the sender must continue to send. However, notice that the receiver algorithm we have advocated involves refraining from acknowledging. Therefore, we certainly do have a situation where improper operation of this algorithm can prevent liveness.

What we must show is that the receiver of the data, if it chooses to refrain from acknowledging, will do so only for a short time, and not forever. The design of the algorithm

described above was intended to achieve precisely this goal: whenever the receiver of data refrained from sending an acknowledgement it was required to set a timer. The only event that was permitted to clear that timer was the receipt of another segment, which essentially reset the timer, and started it going again. Thus, an acknowledgement will be sent as soon as no data has been received. This has precisely the effect desired: if the data flow appears to be disrupted for any reason, the receiver responds by sending an up-to-date acknowledgement. In fact, the receiver algorithm is designed to be more robust than this, for transmission of an acknowledgment is triggered by two events, either a cessation of data or a reduction in the amount of offered window to 50 percent of the actual value. This is the condition which will normally trigger the transmission of this acknowledgement.

Appendix A
Dynamic Calculation of Acknowledgement Delay

The text suggested that when setting a timer to postpone the sending of an acknowledgement, a fixed interval of 200 to 300 milliseconds would work properly in practice. This has not been verified over a wide variety of network delays, and clearly if there is a very slow net which stretches out the intersegment arrival time, a fixed interval will fail. In a sophisticated TCP, which is expected to adjust dynamically (rather than manually) to changing network conditions, it would be appropriate to measure this interval and respond dynamically. The following algorithm, which has been relegated to an Appendix, because it has not been tested, seems sensible. Whenever a segment arrives which does not have the push bit on in it, start a timer, which runs until the next segment arrives. Average these interarrival intervals, using an exponential decay smoothing function tuned to take into account perhaps the last ten or twenty segments that have come in. Occasionally, there will be a long interarrival period, even for a segment which is does not terminate a piece of data being pushed, perhaps because a window has gone to zero or some glitch in the sender or the network has held up the data. Therefore, examine each interarrival interval, and discard it from the smoothing algorithm if it exceeds the current estimate by some amount, perhaps a ratio of two or four times. By rejecting the larger intersegment arrival intervals, one should obtain a smoothed estimate of the interarrival of segments inside a burst. The number need not be exact, since the timer which triggers acknowledgement can add a fairly generous fudge factor to this without causing trouble with the sender's estimate of the retransmission interval, so long as the fudge factor is constant.

RFC-814 Names, Addresses, Ports, and Routes

David D. Clark
MIT Laboratory for Computer Science
July 1982

Introduction

It has been said that the principal function of an operating system is to define a number of different names for the same object, so that it can busy itself keeping track of the relationship between all of the different names. Network protocols seem to have somewhat the same characteristic. In TCP/IP, there are several ways of referring to things. At the human visible interface, there are character string "names" to identify networks, hosts, and services. Host names are translated into network "addresses", 32-bit values that identify the network to which a host is attached, and the location of the host on that net. Service names are translated into a "port identifier", which in TCP is a 16-bit value. Finally, addresses are translated into "routes", which are the sequence of steps a packet must take to reach the specified addresses. Routes show up explicitly in the form of the internet routing options, and also implicitly in the address to route translation tables which all hosts and gateways maintain.

This RFC gives suggestions and guidance for the design of the tables and algorithms necessary to keep track of these various sorts of identifiers inside a host implementation of TCP/IP.

Note: This RFC was written several years before the Domain Naming System was proposed and implemented. Many of the issues raised are addressed by this system which is now a recommended standard of the Internet.

The Scope of the Problem

One of the first questions one can ask about a naming mechanism is how many names one can expect to encounter. In order to answer this, it is necessary to know something about the expected maximum size of the internet. Currently, the internet is fairly small. It contains no more than 25 active networks, and no more than a few hundred hosts. This makes it possible to install tables which exhaustively list all of these elements. However, any implementation undertaken now should be based on an assumption of a much larger internet. The guidelines currently recommended are an upper limit of about 1,000 networks. If we imagine an average number of 25 hosts per net, this would suggest a maximum number of 25,000 hosts. It is quite unclear whether this host estimate is high or low, but even if it is off by several factors of two, the resulting number is still large enough to suggest that current table management strategies are unacceptable. Some fresh techniques will be required to deal with the internet of the future.

RFC-814 Names, Addresses, Ports, and Routes

Names

As the previous section suggests, the internet will eventually have a sufficient number of names that a host cannot have a static table which provides a translation from every name to its associated address. There are several reasons other than sheer size why a host would not wish to have such a table. First, with that many names, we can expect names to be added and deleted at such a rate that an installer might spend all his time just revising the table. Second, most of the names will refer to addresses of machines with which nothing will ever be exchanged. In fact, there may be whole networks with which a particular host will never have any traffic.

To cope with this large and somewhat dynamic environment, the internet is moving from its current position in which a single name table is maintained by the NIC and distributed to all hosts, to a distributed approach in which each network (or group of networks) is responsible for maintaining its own names and providing a "name server" to translate between the names and the addresses in that network. Each host is assumed to store not a complete set of name-address translations, but only a cache of recently used names. When a name is provided by a user for translation to an address, the host will first examine its local cache, and if the name is not found there, will communicate with an appropriate name server to obtain the information, which it may then insert into its cache for future reference.

Unfortunately, the name server mechanism is not totally in place in the internet yet, so for the moment, it is necessary to continue to use the old strategy of maintaining a complete table of all names in every host. Implementors, however, should structure this table in such a way that it is easy to convert later to a name server approach. In particular, a reasonable programming strategy would be to make the name table accessible only through a subroutine interface, rather than by scattering direct references to the table all through the code. In this way, it will be possible, at a later date, to replace the subroutine with one capable of making calls on remote name servers.

A problem which occasionally arises in the ARPANET today is that the information in a local host table is out of date, because a host has moved, and a revision of the host table has not yet been installed from the NIC. In this case, one attempts to connect to a particular host and discovers an unexpected machine at the address obtained from the local table. If a human is directly observing the connection attempt, the error is usually detected immediately. However, for unattended operations such as the sending of queued mail, this sort of problem can lead to a great deal of confusion.

The nameserver scheme will only make this problem worse, if hosts cache locally the address associated with names that have been looked up, because the host has no way of knowing when the address has changed and the cache entry should be removed. To solve this problem, plans are currently under way to define a simple facility by which a host can query a foreign address to determine what name is actually associated with it. SMTP already defines a verification technique based on this approach.

RFC-814 Names, Addresses, Ports, and Routes

Addresses

The IP layer must know something about addresses. In particular, when a datagram is being sent out from a host, the IP layer must decide where to send it on the immediately connected network, based on the internet address. Mechanically, the IP first tests the internet address to see whether the network number of the recipient is the same as the network number of the sender. If so, the packet can be sent directly to the final recipient. If not, the datagram must be sent to a gateway for further forwarding. In this latter case, a second decision must be made, as there may be more than one gateway available on the immediately attached network.

When the internet address format was first specified, 8 bits were reserved to identify the network. Early implementations thus implemented the above algorithm by means of a table with 256 entries, one for each possible net, that specified the gateway of choice for that net, with a special case entry for those nets to which the host was immediately connected. Such tables were sometimes statically filled in, which caused confusion and malfunctions when gateways and networks moved (or crashed).

The current definition of the internet address provides three different options for network numbering, with the goal of allowing a very large number of networks to be part of the internet. Thus, it is no longer possible to imagine having an exhaustive table to select a gateway for any foreign net. Again, current implementations must use a strategy based on a local cache of routing information for addresses currently being used.

The recommended strategy for address to route translation is as follows. When the IP layer receives an outbound datagram for transmission, it extracts the network number from the destination address, and queries its local table to determine whether it knows a suitable gateway to which to send the datagram. If it does, the job is done. (But see RFC 816 on Fault Isolation and Recovery, for recommendations on how to deal with the possible failure of the gateway.) If there is no such entry in the local table, then select any accessible gateway at random, insert that as an entry in the table, and use it to send the packet. Either the guess will be right or wrong. If it is wrong, the gateway to which the packet was sent will return an ICMP redirect message to report that there is a better gateway to reach the net in question. The arrival of this redirect should cause an update of the local table.

The number of entries in the local table should be determined by the maximum number of active connections which this particular host can support at any one time. For a large time sharing system, one might imagine a table with 100 or more entries. For a personal computer being used to support a single user telnet connection, only one address to gateway association need be maintained at once.

The above strategy actually does not completely solve the problem, but only pushes it down one level, where the problem then arises of how a new host, freshly arriving on the internet, finds all of its accessible gateways. Intentionally, this problem is not solved within the internetwork architecture. The reason is that different networks have drastically different strategies for allowing a host to find out about other hosts on its immediate network. Some nets permit a broadcast mechanism. In this case, a host can send out a message and expect an answer back from all of the attached gateways. In other cases, where a particular network is richly provided with tools to support the internet, there may be a special network mechanism which a host can invoke to determine where the gateways are. In other cases, it may be necessary for an installer to manually provide the name of at least one accessible gateway. Once a host has discovered the name of one gateway, it can build up a table of all other available gateways, by keeping track of every gateway that has been reported back to it in an ICMP message.

Advanced Topics in Addressing and Routing

The preceding discussion describes the mechanism required in a minimal implementation, an implementation intended only to provide operational service access today to the various networks that make up the internet. For any host which will participate in future research, as contrasted with service, some additional features are required. These features will also be helpful for service hosts if they wish to obtain access to some of the more exotic networks which will become part of the internet over the next few years. All implementors are urged to at least provide a structure into which these features could be later integrated.

There are several features, either already a part of the architecture or now under development, which are used to modify or expand the relationships between addresses and routes. The IP source route options allow a host to explicitly direct a datagram through a series of gateways to its foreign host. An alternative form of the ICMP redirect packet has been proposed, which would return information specific to a particular destination host, not a destination net. Finally, additional IP options have been proposed to identify particular routes within the internet that are unacceptable. The difficulty with implementing these new features is that the mechanisms do not lie entirely within the bounds of IP. All the mechanisms above are designed to apply to a particular connection, so that their use must be specified at the TCP level. Thus, the interface between IP and the layers above it must include mechanisms to allow passing this information back and forth, and TCP (or any other protocol at this level, such as UDP), must be prepared to store this information. The passing of information between IP and TCP is made more complicated by the fact that some of the information, in particular ICMP packets, may arrive at any time. The normal interface envisioned between TCP and IP is one across which packets can be sent or received. The existence of asynchronous ICMP messages implies that there must be an additional channel between the two, unrelated to the actual sending and receiving of data. (In fact, there are many other ICMP messages which arrive asynchronously and which must be passed from IP up to higher layers. See RFC 816, Fault Isolation and Recovery.)

Source routes are already in use in the internet, and many implementations will wish to be able to take advantage of them. The following sorts of usages should be permitted. First, a user, when initiating a TCP connection, should be able to hand a source route into TCP, which in turn must hand the source route to IP with every outgoing datagram. The user might initially obtain the source route by querying a different sort of name server, which would return a source route instead of an address, or the user may have fabricated the source route manually. A TCP which is listening for a connection, rather than attempting to open one, must be prepared to receive a datagram which contains a IP return route, in which case it must remember this return route, and use it as a source route on all returning datagrams.

RFC-814 Names, Addresses, Ports, and Routes

Ports and Service Identifiers

The IP layer of the architecture contains the address information which specifies the destination host to which the datagram is being sent. In fact, datagrams are not intended just for particular hosts, but for particular agents within a host, processes or other entities that are the actual source and sink of the data. IP performs only a very simple dispatching once the datagram has arrived at the target host, it dispatches it to a particular protocol. It is the responsibility of that protocol handler, for example TCP, to finish dispatching the datagram to the particular connection for which it is destined. This next layer of dispatching is done using "port identifiers", which are a part of the header of the higher level protocol, and not the IP layer.

This two-layer dispatching architecture has caused a problem for certain implementations. In particular, some implementations have wished to put the IP layer within the kernel of the operating system, and the TCP layer as a user domain application program. Strict adherence to this partitioning can lead to grave performance problems, for the datagram must first be dispatched from the kernel to a TCP process, which then dispatches the datagram to its final destination process. The overhead of scheduling this dispatch process can severely limit the achievable throughput of the implementation.

As is discussed in RFC 817, Modularity and Efficiency in Protocol Implementations, this particular separation between kernel and user leads to other performance problems, even ignoring the issue of port level dispatching. However, there is an acceptable shortcut which can be taken to move the higher level dispatching function into the IP layer, if this makes the implementation substantially easier.

In principle, every higher level protocol could have a different dispatching algorithm. The reason for this is discussed below. However, for the protocols involved in the service offering being implemented today, TCP and UDP, the dispatching algorithm is exactly the same, and the port field is located in precisely the same place in the header. Therefore, unless one is interested in participating in further protocol research, there is only one higher level dispatch algorithm. This algorithm takes into account the internet level foreign address, the protocol number, and the local port and foreign port from the higher level protocol header. This algorithm can be implemented as a sort of adjunct to the IP layer implementation, as long as no other higher level protocols are to be implemented. (Actually, the above statement is only partially true, in that the UDP dispatch function is subset of the TCP dispatch function. UDP dispatch depends only protocol number and local port. However, there is an occasion within TCP when this exact same subset comes into play, when a process wishes to listen for a connection from any foreign host. Thus, the range of mechanisms necessary to support TCP dispatch are also sufficient to support precisely the UDP requirement.)

The decision to remove port level dispatching from IP to the higher level protocol has been questioned by some implementors. It has been argued that if all of the address structure were part of the IP layer, then IP could do all of the packet dispatching function within the host, which would lead to a simpler modularity. Three problems were identified with this. First, not all protocol implementors could agree on the size of the port identifier. TCP selected a fairly short port identifier, 16 bits, to reduce header size. Other protocols being designed, however, wanted a larger port identifier, perhaps 32 bits, so that the port identifier, if properly selected, could be considered probabilistically unique. Thus, constraining the port id to one particular IP level mechanism would prevent certain fruitful lines of research. Second, ports serve a special function in addition to datagram delivery: certain port numbers are reserved to identify particular services. Thus, TCP port 23 is the remote login service. If ports were implemented at the IP level, then the assignment of well

known ports could not be done on a protocol basis, but would have to be done in a centralized manner for all of the IP architecture. Third, IP was designed with a very simple layering role: IP contained exactly those functions that the gateways must understand. If the port idea had been made a part of the IP layer, it would have suggested that gateways needed to know about ports, which is not the case.

There are, of course, other ways to avoid these problems. In particular, the "well-known port" problem can be solved by devising a second mechanism, distinct from port dispatching, to name well-known ports. Several protocols have settled on the idea of including, in the packet which sets up a connection to a particular service, a more general service descriptor, such as a character string field. These special packets, which are requesting connection to a particular service, are routed on arrival to a special server, sometimes called a "rendezvous server", which examines the service request, selects a random port which is to be used for this instance of the service, and then passes the packet along to the service itself to commence the interaction.

For the internet architecture, this strategy had the serious flaw that it presumed all protocols would fit into the same service paradigm: an initial setup phase, which might contain a certain overhead such as indirect routing through a rendezvous server, followed by the packets of the interaction itself, which would flow directly to the process providing the service. Unfortunately, not all high level protocols in internet were expected to fit this model. The best example of this is isolated datagram exchange using UDP. The simplest exchange in UDP is one process sending a single datagram to another. Especially on a local net, where the net related overhead is very low, this kind of simple single datagram interchange can be extremely efficient, with very low overhead in the hosts. However, since these individual packets would not be part of an established connection, if IP supported a strategy based on a rendezvous server and service descriptors, every isolated datagram would have to be routed indirectly in the receiving host through the rendezvous server, which would substantially increase the overhead of processing, and every datagram would have to carry the full service request field, which would increase the size of the packet header.

In general, if a network is intended for "virtual circuit service", or things similar to that, then using a special high overhead mechanism for circuit setup makes sense. However, current directions in research are leading away from this class of protocol, so once again the architecture was designed not to preclude alternative protocol structures. The only rational position was that the particular dispatching strategy used should be part of the higher level protocol design, not the IP layer.

This same argument about circuit setup mechanisms also applies to the design of the IP address structure. Many protocols do not transmit a full address field as part of every packet, but rather transmit a short identifier which is created as part of a circuit setup from source to destination. If the full address needs to be carried in only the first packet of a long exchange, then the overhead of carrying a very long address field can easily be justified. Under these circumstances, one can create truly extravagant address fields, which are capable of extending to address almost any conceivable entity. However, this strategy is useable only in a virtual circuit net, where the packets being transmitted are part of a established sequence, otherwise this large extravagant address must be transported on every packet. Since Internet explicitly rejected this restriction on the architecture, it was necessary to come up with an address field that was compact enough to be sent in every datagram, but general enough to correctly route the datagram through the catanet without a previous setup phase. The IP address of 32 bits is the compromise that results. Clearly it requires a substantial amount of shoehorning to address all of the interesting places in the universe with only 32 bits. On the other hand, had the address field become much bigger, IP would have been susceptible to another criticism, which is that the header had grown unworkably large. Again, the fundamental design decision was that the protocol be designed in such a way that it supported research in new and different sorts of protocol

architectures.

There are some limited restrictions imposed by the IP design on the port mechanism selected by the higher level process. In particular, when a packet goes awry somewhere on the internet, the offending packet is returned, along with an error indication, as part of an ICMP packet. An ICMP packet returns only the IP layer, and the next 64 bits of the original datagram. Thus, any higher level protocol which wishes to sort out from which port a particular offending datagram came must make sure that the port information is contained within the first 64 bits of the next level header. This also means, in most cases, that it is possible to imagine, as part of the IP layer, a port dispatch mechanism which works by masking and matching on the first 64 bits of the incoming higher level header.

RFC-815 IP Datagram Reassembly Algorithms

David D. Clark
MIT Laboratory for Computer Science
July 1982

RFC-815 IP Datagram Reassembly Algorithms

Introduction

One of the mechanisms of IP is fragmentation and reassembly. Under certain circumstances, a datagram originally transmitted as a single unit will arrive at its final destination broken into several fragments. The IP layer at the receiving host must accumulate these fragments until enough have arrived to completely reconstitute the original datagram. The specification document for IP gives a complete description of the reassembly mechanism, and contains several examples. It also provides one possible algorithm for reassembly, based on keeping track of arriving fragments in a vector of bits. This document describes an alternate approach which should prove more suitable in some machines.

A superficial examination of the reassembly process may suggest that it is rather complicated. First, it is necessary to keep track of all the fragments, which suggests a small bookkeeping job. Second, when a new fragment arrives, it may combine with the existing fragments in a number of different ways. It may precisely fill the space between two fragments, or it may overlap with existing fragments, or completely duplicate existing fragments, or partially fill a space between two fragments without abutting either of them. Thus, it might seem that the reassembly process might involve designing a fairly complicated algorithm that tests for a number of different options.

In fact, the process of reassembly is extremely simple. This document describes a way of dealing with reassembly which reduces the bookkeeping problem to a minimum, which requires for storage only one buffer equal in size to the final datagram being reassembled, which can reassemble a datagram from any number of fragments arriving in any order with any possible pattern of overlap and duplication, and which is appropriate for almost any sort of operating system.

The reader should consult the IP specification document to be sure that he is completely familiar with the general concept of reassembly, and the particular header fields and vocabulary used to describe the process.

The Algorithm

In order to define this reassembly algorithm, it is necessary to define some terms. A partially reassembled datagram consists of certain sequences of octets that have already arrived, and certain areas still to come. We will refer to these missing areas as "holes". Each hole can be characterized by two numbers, hole.first, the number of the first octet in the hole, and hole.last, the number of the last octet in the hole. This pair of numbers we will call the "hole descriptor", and we will assume that all of the hole descriptors for a particular datagram are gathered together in the "hole descriptor list".

The general form of the algorithm is as follows. When a new fragment of the datagram arrives, it will possibly fill in one or more of the existing holes. We will examine each of the entries in the hole descriptor list to see whether the hole in question is eliminated by this incoming fragment. If so, we will delete that entry from the list. Eventually, a fragment will arrive which eliminates every entry from the list. At this point, the datagram has been completely reassembled and can be passed to higher protocol levels for further processing.

The algorithm will be described in two phases. In the first part, we will show the sequence of steps which are executed when a new fragment arrives, in order to determine whether or not any of the existing holes are filled by the new fragment. In the second part of this description, we will show a ridiculously simple algorithm for management of the hole descriptor list.

RFC-815 IP Datagram Reassembly Algorithms

Fragment Processing Algorithm

An arriving fragment can fill any of the existing holes in a number of ways. Most simply, it can completely fill a hole. Alternatively, it may leave some remaining space at either the beginning or the end of an existing hole. Or finally, it can lie in the middle of an existing hole, breaking the hole in half and leaving a smaller hole at each end. Because of these possibilities, it might seem that a number of tests must be made when a new fragment arrives, leading to a rather complicated algorithm. In fact, if properly expressed, the algorithm can compare each hole to the arriving fragment in only four tests.

We start the algorithm when the earliest fragment of the datagram arrives. We begin by creating an empty data buffer area and putting one entry in its hole descriptor list, the entry which describes the datagram as being completely missing. In this case, hole.first equals zero, and hole.last equals infinity. (Infinity is presumably implemented by a very large integer, greater than 576, of the implementor's choice.) The following eight steps are then used to insert each of the arriving fragments into the buffer area where the complete datagram is being built up. The arriving fragment is described by fragment.first, the first octet of the fragment, and fragment.last, the last octet of the fragment.

1. Select the next hole descriptor from the hole descriptor list. If there are no more entries, go to step eight.
2. If fragment.first is greater than hole.last, go to step one.
3. If fragment.last is less than hole.first, go to step one.

- (If either step two or step three is true, then the newly arrived fragment does not overlap with the hole in any way, so we need pay no further attention to this hole. We return to the beginning of the algorithm where we select the next hole for examination.)

4. Delete the current entry from the hole descriptor list.

- (Since neither step two nor step three was true, the newly arrived fragment does interact with this hole in some way. Therefore, the current descriptor will no longer be valid. We will destroy it, and in the next two steps we will determine whether or not it is necessary to create any new hole descriptors.)

5. If fragment.first is greater than hole.first, then create a new hole descriptor "new_hole" with new_hole.first equal to hole.first, and new_hole.last equal to fragment.first minus one.

- (If the test in step five is true, then the first part of the original hole is not filled by this fragment. We create a new descriptor for this smaller hole.)

6. If fragment.last is less than hole.last and fragment.more fragments is true, then create a new hole descriptor "new_hole", with new_hole.first equal to fragment.last plus one and new_hole.last equal to hole.last.

- (This test is the mirror of step five with one additional feature. Initially, we did not know how long the reassembled datagram would be, and therefore we created a hole reaching from zero to infinity. Eventually, we will receive the last fragment of the datagram. At this point, that hole descriptor which reaches from the last octet of the buffer to infinity can be discarded. The fragment which contains the last fragment indicates this fact by a flag in the internet header called "more fragments". The test of this bit in this statement prevents us from creating a descriptor for the

unneeded hole which describes the space from the end of the datagram to infinity.)

7. Go to step one.
8. If the hole descriptor list is now empty, the datagram is now complete. Pass it on to the higher level protocol processor for further handling. Otherwise, return.

Part Two: Managing the Hole Descriptor List

The main complexity in the eight step algorithm above is not performing the arithmetical tests, but in adding and deleting entries from the hole descriptor list. One could imagine an implementation in which the storage management package was many times more complicated than the rest of the algorithm, since there is no specified upper limit on the number of hole descriptors which will exist for a datagram during reassembly. There is a very simple way to deal with the hole descriptors, however. Just put each hole descriptor in the first octets of the hole itself. Note that by the definition of the reassembly algorithm, the minimum size of a hole is eight octets. To store hole.first and hole.last will presumably require two octets each. An additional two octets will be required to thread together the entries on the hole descriptor list. This leaves at least two more octets to deal with implementation idiosyncrasies.

There is only one obvious pitfall to this storage strategy. One must execute the eight step algorithm above before copying the data from the fragment into the reassembly buffer. If one were to copy the data first, it might smash one or more hole descriptors. Once the algorithm above has been run, any hole descriptors which are about to be smashed have already been rendered obsolete.

Loose Ends

Scattering the hole descriptors throughout the reassembly buffer itself requires that they be threaded onto some sort of list so that they can be found. This in turn implies that there must be a pointer to the head of the list. In many cases, this pointer can be stored in some sort of descriptor block which the implementation associates with each reassembly buffer. If no such storage is available, a dirty but effective trick is to store the head of the list in a part of the internet header in the reassembly buffer which is no longer needed. An obvious location is the checksum field.

When the final fragment of the datagram arrives, the packet length field in the internet header should be filled in.

Options

The preceding description made one unacceptable simplification. It assumed that there were no internet options associated with the datagram being reassembled. The difficulty with options is that until one receives the first fragment of the datagram, one cannot tell how big the internet header will be. This is because, while certain options are copied identically into every fragment of a datagram, other options, such as "record route", are put in the first fragment only. (The "first fragment" is the fragment containing octet zero of the original datagram.)

Until one knows how big the internet header is, one does not know where to copy the data from each fragment into the reassembly buffer. If the earliest fragment to arrive happens to be the first fragment, then this is no problem. Otherwise, there are two solutions. First, one can leave space in the reassembly buffer for the maximum possible internet header. In fact, the maximum size is not very large, 64 octets. Alternatively, one can simply gamble that the first fragment will contain no options. If, when the first fragment finally arrives, there are options, one can then shift the data in the buffer a sufficient distance for allow for them. The only peril in copying the data is that one will trash the pointers that thread the hole descriptors together. It is easy to see how to untrash the pointers.

The source and record route options have the interesting feature that, since different fragments can follow different paths, they may arrive with different return routes recorded in different fragments. Normally, this is more information than the receiving Internet module needs. The specified procedure is to take the return route recorded in the first fragment and ignore the other versions.

The Complete Algorithm

In addition to the algorithm described above there are two parts to the reassembly process. First, when a fragment arrives, it is necessary to find the reassembly buffer associated with that fragment. This requires some mechanism for searching all the existing reassembly buffers. The correct reassembly buffer is identified by an equality of the following fields: the foreign and local internet address, the protocol ID, and the identification field.

The final part of the algorithm is some sort of timer based mechanism which decrements the time to live field of each partially reassembled datagram, so that incomplete datagrams which have outlived their usefulness can be detected and deleted. One can either create a daemon which comes alive once a second and decrements the field in each datagram by one, or one can read the clock when each first fragment arrives, and queue some sort of timer call, using whatever system mechanism is appropriate, to reap the datagram when its time has come.

An implementation of the complete algorithm comprising all these parts was constructed in BCPL as a test. The complete algorithm took less than one and one-half pages of listing, and generated approximately 400 nova machine instructions. That portion of the algorithm actually involved with management of hole descriptors is about 20 lines of code.

The version of the algorithm described here is actually a simplification of the author's original version, thanks to an insightful observation by Elizabeth Martin at MIT.

RFC-816 Fault Isolation and Recovery

David D. Clark
MIT Laboratory for Computer Science
July 1982

Introduction

Occasionally, a network or a gateway will go down, and the sequence of hops which the packet takes from source to destination must change. Fault isolation is that action which hosts and gateways collectively take to determine that something is wrong; fault recovery is the identification and selection of an alternative route which will serve to reconnect the source to the destination. In fact, the gateways perform most of the functions of fault isolation and recovery. There are, however, a few actions which hosts must take if they wish to provide a reasonable level of service. This document describes the portion of fault isolation and recovery which is the responsibility of the host.

What Gateways Do

Gateways collectively implement an algorithm which identifies the best route between all pairs of networks. They do this by exchanging packets which contain each gateway's latest opinion about the operational status of its neighbor networks and gateways. Assuming that this algorithm is operating properly, one can expect the gateways to go through a period of confusion immediately after some network or gateway has failed, but one can assume that once a period of negotiation has passed, the gateways are equipped with a consistent and correct model of the connectivity of the internet. At present this period of negotiation may actually take several minutes, and many TCP implementations time out within that period, but it is a design goal of the eventual algorithm that the gateway should be able to reconstruct the topology quickly enough that a TCP connection should be able to survive a failure of the route.

Host Algorithm for Fault Recovery

Since the gateways always attempt to have a consistent and correct model of the internetwork topology, the host strategy for fault recovery is very simple. Whenever the host feels that something is wrong, it asks the gateway for advice, and, assuming the advice is forthcoming, it believes the advice completely. The advice will be wrong only during the transient period of negotiation, which immediately follows an outage, but will otherwise be reliably correct.

In fact, it is never necessary for a host to explicitly ask a gateway for advice, because the gateway will provide it as appropriate. When a host sends a datagram to some distant net, the host should be prepared to receive back either of two advisory messages which the gateway may send. The ICMP "redirect" message indicates that the gateway to which the host sent the datagram is not longer the best gateway to reach the net in question. The gateway will have forwarded the datagram, but the host should revise its routing table to have a different immediate address for this net. The ICMP "destination unreachable" message indicates that as a result of an outage, it is currently impossible to reach the addressed net or host in any manner. On receipt of this message, a host can either abandon the connection immediately without any further retransmission, or resend slowly to see if the fault is corrected in reasonable time.

If a host could assume that these two ICMP messages would always arrive when something was amiss in the network, then no other action on the part of the host would be required in order maintain its tables in an optimal condition. Unfortunately, there are two circumstances under which the messages will not arrive properly. First, during the transient following a failure, error messages may arrive that do not correctly represent the state of the world. Thus, hosts must take an isolated error message with some scepticism. (This transient period is discussed more fully below.) Second, if the host has been sending datagrams to a particular gateway, and that gateway itself crashes, then all the other gateways in the internet will reconstruct the topology, but the gateway in question will still be down, and therefore cannot provide any advice back to the host. As long as the host continues to direct datagrams at this dead gateway, the datagrams will simply vanish off the face of the earth, and nothing will come back in return. Hosts must detect this failure.

If some gateway many hops away fails, this is not of concern to the host, for then the discovery of the failure is the responsibility of the immediate neighbor gateways, which will perform this action in a manner invisible to the host. The problem only arises if the very first gateway, the one to which the host is immediately sending the datagrams, fails. We thus identify one single task which the host must perform as its part of fault isolation in the internet: the host must use some strategy to detect that a gateway to which it is sending datagrams is dead.

Let us assume for the moment that the host implements some algorithm to detect failed gateways; we will return later to discuss what this algorithm might be. First, let us consider what the host should do when it has determined that a gateway is down. In fact, with the exception of one small problem, the action the host should take is extremely simple. The host should select some other gateway, and try sending the datagram to it. Assuming that gateway is up, this will either produce correct results, or some ICMP advice. Since we assume that, ignoring temporary periods immediately following an outage, any gateway is capable of giving correct advice, once the host has received advice from any gateway, that host is in as good a condition as it can hope to be.

There is always the unpleasant possibility that when the host tries a different gateway, that gateway too will be down. Therefore, whatever algorithm the host uses to detect a dead gateway must continuously be applied, as the host tries every gateway in turn that it knows about.

The only difficult part of this algorithm is to specify the means by which the host maintains the table of all of the gateways to which it has immediate access. Currently, the specification of the internet protocol does not architect any message by which a host can ask to be supplied with such a table. The reason is that different networks may provide very different mechanisms by which this table can be filled in. For example, if the net is a broadcast net, such as an ethernet or a ringnet, every gateway may simply broadcast such a table from time to time, and the host need do nothing but listen to obtain the required information. Alternatively, the network may provide the mechanism of logical addressing, by which a whole set of machines can be provided with a single group address, to which a request can be sent for assistance. Failing those two schemes, the host can build up its table of neighbor gateways by remembering all the gateways from which it has ever received a message. Finally, in certain cases, it may be necessary for this table, or at least the initial entries in the table, to be constructed manually by a manager or operator at the site. In cases where the network in question provides absolutely no support for this kind of host query, at least some manual intervention will be required to get started, so that the host can find out about at least one gateway.

Host Algorithms for Fault Isolation

We now return to the question raised above. What strategy should the host use to detect that it is talking to a dead gateway, so that it can know to switch to some other gateway in the list. In fact, there are several algorithms which can be used. All are reasonably simple to implement, but they have very different implications for the overhead on the host, the gateway, and the network. Thus, to a certain extent, the algorithm picked must depend on the details of the network and of the host.

Network Level Detection

Many networks, particularly the Arpanet, perform precisely the required function internal to the network. If a host sends a datagram to a dead gateway on the Arpanet, the network will return a "host dead" message, which is precisely the information the host needs to know in order to switch to another gateway. Some early implementations of Internet on the Arpanet threw these messages away. That is an exceedingly poor idea.

Continuous Polling

The ICMP protocol provides an echo mechanism by which a host may solicit a response from a gateway. A host could simply send this message at a reasonable rate, to assure itself continuously that the gateway was still up. This works, but, since the message must be sent fairly often to detect a fault in a reasonable time, it can imply an unbearable overhead on the host itself, the network, and the gateway. This strategy is prohibited except where a specific analysis has indicated that the overhead is tolerable.

Triggered Polling

If the use of polling could be restricted to only those times when something seemed to be wrong, then the overhead would be bearable. Provided that one can get the proper advice from one's higher level protocols, it is possible to implement such a strategy. For example, one could program the TCP level so that whenever it retransmitted a segment more than once, it sent a hint down to the IP layer which triggered polling. This strategy does not have excessive overhead, but does have the problem that the host may be somewhat slow to respond to an error, since only after polling has started will the host be able to confirm that something has gone wrong, and by then the TCP above may have already timed out.

Both forms of polling suffer from a minor flaw. Hosts as well as gateways respond to ICMP echo messages. Thus, polling cannot be used to detect the error that a foreign address thought to be a gateway is actually a host. Such a confusion can arise if the physical addresses of machines are rearranged.

Triggered Reselection

There is a strategy which makes use of a hint from a higher level, as did the previous strategy, but which avoids polling altogether. Whenever a higher level complains that the service seems to be defective, the Internet layer can pick the next gateway from the list of available gateways, and switch to it. Assuming that this gateway is up, no real harm can come of this decision, even if it was wrong, for the worst that will happen is a redirect message which instructs the host to return to the gateway originally being used. If, on the other hand, the original gateway was indeed down, then this immediately provides a new route, so the period of time until recovery is shortened. This last strategy seems particularly clever, and is probably the most generally suitable for those cases where the network itself does not provide fault isolation. (Regretably, I have forgotten who suggested this idea to me. It is not my invention.)

Higher Level Fault Detection

The previous discussion has concentrated on fault detection and recovery at the IP layer. This section considers what the higher layers such as TCP should do.

TCP has a single fault recovery action; it repeatedly retransmits a segment until either it gets an acknowledgement or its connection timer expires. As discussed above, it may use retransmission as an event to trigger a request for fault recovery to the IP layer. In the other direction, information may flow up from IP, reporting such things as ICMP Destination Unreachable or error messages from the attached network. The only subtle question about TCP and faults is what TCP should do when such an error message arrives or its connection timer expires.

The TCP specification discusses the timer. In the description of the open call, the timeout is described as an optional value that the client of TCP may specify; if any segment remains unacknowledged for this period, TCP should abort the connection. The default for the timeout is 30 seconds. Early TCPs were often implemented with a fixed timeout interval, but this did not work well in practice, as the following discussion may suggest.

Clients of TCP can be divided into two classes: those running on immediate behalf of a human, such as Telnet, and those supporting a program, such as a mail sender. Humans require a sophisticated response to errors. Depending on exactly what went wrong, they may want to abandon the connection at once, or wait for a long time to see if things get better. Programs do not have this human impatience, but also lack the power to make complex decisions based on details of the exact error condition. For them, a simple timeout is reasonable.

Based on these considerations, at least two modes of operation are needed in TCP. One, for programs, abandons the connection without exception if the TCP timer expires. The other mode, suitable for people, never abandons the connection on its own initiative, but reports to the layer above when the timer expires. Thus, the human user can see error messages coming from all the relevant layers, TCP and ICMP, and can request TCP to abort as appropriate. This second mode requires that TCP be able to send an asynchronous message up to its client to report the timeout, and it requires that error messages arriving at lower layers similarly flow up through TCP.

At levels above TCP, fault detection is also required. Either of the following can happen. First, the foreign client of TCP can fail, even though TCP is still running, so data is still acknowledged and the timer never expires. Alternatively, the communication path can fail, without the TCP timer going off, because the local client has no data to send. Both of these have caused trouble.

Sending mail provides an example of the first case. When sending mail using SMTP, there is an SMTP level acknowledgement that is returned when a piece of mail is successfully delivered. Several early mail receiving programs would crash just at the point where they had received all of the mail text (so TCP did not detect a timeout due to outstanding unacknowledged data) but before the mail was acknowledged at the SMTP level. This failure would cause early mail senders to wait forever for the SMTP level acknowledgement. The obvious cure was to set a timer at the SMTP level, but the first attempt to do this did not work, for there was no simple way to select the timer interval. If the interval selected was short, it expired in normal operational when sending a large file to a slow host. An interval of many minutes was needed to prevent false timeouts, but that meant that failures were detected only very slowly. The current solution in several mailers is to pick a timeout interval proportional to the size of the

message.

Server telnet provides an example of the other kind of failure. It can easily happen that the communications link can fail while there is no traffic flowing, perhaps because the user is thinking. Eventually, the user will attempt to type something, at which time he will discover that the connection is dead and abort it. But the host end of the connection, having nothing to send, will not discover anything wrong, and will remain waiting forever. In some systems there is no way for a user in a different process to destroy or take over such a hanging process, so there is no way to recover.

One solution to this would be to have the host server telnet query the user end now and then, to see if it is still up. (Telnet does not have an explicit query feature, but the host could negotiate some unimportant option, which should produce either agreement or disagreement in return.) The only problem with this is that a reasonable sample interval, if applied to every user on a large system, can generate an unacceptable amount of traffic and system overhead. A smart server telnet would use this query only when something seems wrong, perhaps when there had been no user activity for some time.

In both these cases, the general conclusion is that client level error detection is needed, and that the details of the mechanism are very dependent on the application. Application programmers must be made aware of the problem of failures, and must understand that error detection at the TCP or lower level cannot solve the whole problem for them.

Knowing When to Give Up

It is not obvious, when error messages such as ICMP Destination Unreachable arrive, whether TCP should abandon the connection. The reason that error messages are difficult to interpret is that, as discussed above, after a failure of a gateway or network, there is a transient period during which the gateways may have incorrect information, so that irrelevant or incorrect error messages may sometimes return. An isolated ICMP Destination Unreachable may arrive at a host, for example, if a packet is sent during the period when the gateways are trying to find a new route. To abandon a TCP connection based on such a message arriving would be to ignore the valuable feature of the Internet that for many internal failures it reconstructs its function without any disruption of the end points.

But if failure messages do not imply a failure, what are they for? In fact, error messages serve several important purposes. First, if they arrive in response to opening a new connection, they probably are caused by opening the connection improperly (e.g., to a non-existent address) rather than by a transient network failure. Second, they provide valuable information, after the TCP timeout has occurred, as to the probable cause of the failure. Finally, certain messages, such as ICMP Parameter Problem, imply a possible implementation problem. In general, error messages give valuable information about what went wrong, but are not to be taken as absolutely reliable. A general alerting mechanism, such as the TCP timeout discussed above, provides a good indication that whatever is wrong is a serious condition, but without the advisory messages to augment the timer, there is no way for the client to know how to respond to the error. The combination of the timer and the advice from the error messages provide a reasonable set of facts for the client layer to have. It is important that error messages from all layers be passed up to the client module in a useful and consistent way.

RFC-817 Modularity and Efficiency in Protocol Implementation

David D. Clark
MIT Laboratory for Computer Science
July 1982

RFC-817 Modularity and Efficiency in Protocol Implementation

Introduction

Many protocol implementers have made the unpleasant discovery that their packages do not run quite as fast as they had hoped. The blame for this widely observed problem has been attributed to a variety of causes, ranging from details in the design of the protocol to the underlying structure of the host operating system. This RFC will discuss some of the commonly encountered reasons why protocol implementations seem to run slowly.

Experience suggests that one of the most important factors in determining the performance of an implementation is the manner in which that implementation is modularized and integrated into the host operating system. For this reason, it is useful to discuss the question of how an implementation is structured at the same time that we consider how it will perform. In fact, this RFC will argue that modularity is one of the chief villains in attempting to obtain good performance, so that the designer is faced with a delicate and inevitable tradeoff between good structure and good performance. Further, the single factor which most strongly determines how well this conflict can be resolved is not the protocol but the operating system.

RFC-817 Modularity and Efficiency in Protocol Implementation

Efficiency Considerations

There are many aspects to efficiency. One aspect is sending data at minimum transmission cost, which is a critical aspect of common carrier communications, if not in local area network communications. Another aspect is sending data at a high rate, which may not be possible at all if the net is very slow, but which may be the one central design constraint when taking advantage of a local net with high raw bandwidth. The final consideration is doing the above with minimum expenditure of computer resources. This last may be necessary to achieve high speed, but in the case of the slow net may be important only in that the resources used up, for example cpu cycles, are costly or otherwise needed. It is worth pointing out that these different goals often conflict; for example it is often possible to trade off efficient use of the computer against efficient use of the network. Thus, there may be no such thing as a successful general purpose protocol implementation.

The simplest measure of performance is throughput, measured in bits per second. It is worth doing a few simple computations in order to get a feeling for the magnitude of the problems involved. Assume that data is being sent from one machine to another in packets of 576 bytes, the maximum generally acceptable internet packet size. Allowing for header overhead, this packet size permits 4288 bits in each packet. If a useful throughput of 10,000 bits per second is desired, then a data bearing packet must leave the sending host about every 430 milliseconds, a little over two per second. This is clearly not difficult to achieve. However, if one wishes to achieve 100 kilobits per second throughput, the packet must leave the host every 43 milliseconds, and to achieve one megabit per second, which is not at all unreasonable on a high-speed local net, the packets must be spaced no more than 4. 3 milliseconds.

These latter numbers are a slightly more alarming goal for which to set one's sights. Many operating systems take a substantial fraction of a millisecond just to service an interrupt. If the protocol has been structured as a process, it is necessary to go through a process scheduling before the protocol code can even begin to run. If any piece of a protocol package or its data must be fetched from disk, real time delays of between 30 to 100 milliseconds can be expected. If the protocol must compete for cpu resources with other processes of the system, it may be necessary to wait a scheduling quantum before the protocol can run. Many systems have a scheduling quantum of 100 milliseconds or more. Considering these sorts of numbers, it becomes immediately clear that the protocol must be fitted into the operating system in a thorough and effective manner if any like reasonable throughput is to be achieved.

There is one obvious conclusion immediately suggested by even this simple analysis. Except in very special circumstances, when many packets are being processed at once, the cost of processing a packet is dominated by factors, such as cpu scheduling, which are independent of the packet size. This suggests two general rules which any implementation ought to obey. First, send data in large packets. Obviously, if processing time per packet is a constant, then throughput will be directly proportional to the packet size. Second, never send an unneeded packet. Unneeded packets use up just as many resources as a packet full of data, but perform no useful function. RFC 813, "Window and Acknowledgement Strategy in TCP", discusses one aspect of reducing the number of packets sent per useful data byte. This document will mention other attacks on the same problem.

The above analysis suggests that there are two main parts to the problem of achieving good protocol performance. The first has to do with how the protocol implementation is integrated into the host operating system. The second has to do with how the protocol package itself is organized internally. This document will consider each of these topics in turn.

The Protocol vs. the Operating System

There are normally three reasonable ways in which to add a protocol to an operating system. The protocol can be in a process that is provided by the operating system, or it can be part of the kernel of the operating system itself, or it can be put in a separate communications processor or front end machine. This decision is strongly influenced by details of hardware architecture and operating system design; each of these three approaches has its own advantages and disadvantages.

The "process" is the abstraction which most operating systems use to provide the execution environment for user programs. A very simple path for implementing a protocol is to obtain a process from the operating system and implement the protocol to run in it. Superficially, this approach has a number of advantages. Since modifications to the kernel are not required, the job can be done by someone who is not an expert in the kernel structure. Since it is often impossible to find somebody who is experienced both in the structure of the operating system and the structure of the protocol, this path, from a management point of view, is often extremely appealing. Unfortunately, putting a protocol in a process has a number of disadvantages, related to both structure and performance. First, as was discussed above, process scheduling can be a significant source of real-time delay. There is not only the actual cost of going through the scheduler, but the problem that the operating system may not have the right sort of priority tools to bring the process into execution quickly whenever there is work to be done.

Structurally, the difficulty with putting a protocol in a process is that the protocol may be providing services, for example support of data streams, which are normally obtained by going to special kernel entry points. Depending on the generality of the operating system, it may be impossible to take a program which is accustomed to reading through a kernel entry point, and redirect it so it is reading the data from a process. The most extreme example of this problem occurs when implementing server telnet. In almost all systems, the device handler for the locally attached teletypes is located inside the kernel, and programs read and write from their teletype by making kernel calls. If server telnet is implemented in a process, it is then necessary to take the data streams provided by server telnet and somehow get them back down inside the kernel so that they mimic the interface provided by local teletypes. It is usually the case that special kernel modification is necessary to achieve this structure, which somewhat defeats the benefit of having removed the protocol from the kernel in the first place.

Clearly, then, there are advantages to putting the protocol package in the kernel. Structurally, it is reasonable to view the network as a device, and device drivers are traditionally contained in the kernel. Presumably, the problems associated with process scheduling can be sidesteped, at least to a certain extent, by placing the code inside the kernel. And it is obviously easier to make the server telnet channels mimic the local teletype channels if they are both realized in the same level in the kernel.

However, implementation of protocols in the kernel has its own set of pitfalls. First, network protocols have a characteristic which is shared by almost no other device: they require rather complex actions to be performed as a result of a timeout. The problem with this requirement is that the kernel often has no facility by which a program can be brought into execution as a result of the timer event. What is really needed, of course, is a special sort of process inside the kernel. Most systems lack this mechanism. Failing that, the only execution mechanism available is to run at interrupt time.

There are substantial drawbacks to implementing a protocol to run at interrupt time. First, the actions performed may be somewhat complex and time consuming, compared to the

maximum amount of time that the operating system is prepared to spend servicing an interrupt. Problems can arise if interrupts are masked for too long. This is particularly bad when running as a result of a clock interrupt, which can imply that the clock interrupt is masked. Second, the environment provided by an interrupt handler is usually extremely primitive compared to the environment of a process. There are usually a variety of system facilities which are unavailable while running in an interrupt handler. The most important of these is the ability to suspend execution pending the arrival of some event or message. It is a cardinal rule of almost every known operating system that one must not invoke the scheduler while running in an interrupt handler. Thus, the programmer who is forced to implement all or part of his protocol package as an interrupt handler must be the best sort of expert in the operating system involved, and must be prepared for development sessions filled with obscure bugs which crash not just the protocol package but the entire operating system.

A final problem with processing at interrupt time is that the system scheduler has no control over the percentage of system time used by the protocol handler. If a large number of packets arrive, from a foreign host that is either malfunctioning or fast, all of the time may be spent in the interrupt handler, effectively killing the system.

There are other problems associated with putting protocols into an operating system kernel. The simplest problem often encountered is that the kernel address space is simply too small to hold the piece of code in question. This is a rather artificial sort of problem, but it is a severe problem none the less in many machines. It is an appallingly unpleasant experience to do an implementation with the knowledge that for every byte of new feature put in one must find some other byte of old feature to throw out. It is hopeless to expect an effective and general implementation under this kind of constraint. Another problem is that the protocol package, once it is thoroughly entwined in the operating system, may need to be redone every time the operating system changes. If the protocol and the operating system are not maintained by the same group, this makes maintenance of the protocol package a perpetual headache.

The third option for protocol implementation is to take the protocol package and move it outside the machine entirely, on to a separate processor dedicated to this kind of task. Such a machine is often described as a communications processor or a front-end processor. There are several advantages to this approach. First, the operating system on the communications processor can be tailored for precisely this kind of task. This makes the job of implementation much easier. Second, one does not need to redo the task for every machine to which the protocol is to be added. It may be possible to reuse the same front-end machine on different host computers. Since the task need not be done as many times, one might hope that more attention could be paid to doing it right. Given a careful implementation in an environment which is optimized for this kind of task, the resulting package should turn out to be very efficient. Unfortunately, there are also problems with this approach. There is, of course, a financial problem associated with buying an additional computer. In many cases, this is not a problem at all since the cost is negligible compared to what the programmer would cost to do the job in the mainframe itself. More fundamentally, the communications processor approach does not completely sidestep any of the problems raised above. The reason is that the communications processor, since it is a separate machine, must be attached to the mainframe by some mechanism. Whatever that mechanism, code is required in the mainframe to deal with it. It can be argued that the program to deal with the communications processor is simpler than the program to implement the entire protocol package. Even if that is so, the communications processor interface package is still a protocol in nature, with all of the same structural problems. Thus, all of the issues raised above must still be faced. In addition to those problems, there are some other, more subtle problems associated with an outboard implementation of a protocol. We will return to these problems later.

There is a way of attaching a communications processor to a mainframe host which

sidesteps all of the mainframe implementation problems, which is to use some preexisting interface on the host machine as the port by which a communications processor is attached. This strategy is often used as a last stage of desperation when the software on the host computer is so intractable that it cannot be changed in any way. Unfortunately, it is almost inevitably the case that all of the available interfaces are totally unsuitable for this purpose, so the result is unsatisfactory at best. The most common way in which this form of attachment occurs is when a network connection is being used to mimic local teletypes. In this case, the front-end processor can be attached to the mainframe by simply providing a number of wires out of the front-end processor, each corresponding to a connection, which are plugged into teletype ports on the mainframe computer. (Because of the appearance of the physical configuration which results from this arrangement, Michael Padlipsky has described this as the "milking machine" approach to computer networking.) This strategy solves the immediate problem of providing remote access to a host, but it is extremely inflexible. The channels being provided to the host are restricted by the host software to one purpose only, remote login. It is impossible to use them for any other purpose, such as file transfer or sending mail, so the host is integrated into the network environment in an extremely limited and inflexible manner. If this is the best that can be done, then it should be tolerated. Otherwise, implementors should be strongly encouraged to take a more flexible approach.

Protocol Layering

The previous discussion suggested that there was a decision to be made as to where a protocol ought to be implemented. In fact, the decision is much more complicated than that, for the goal is not to implement a single protocol, but to implement a whole family of protocol layers, starting with a device driver or local network driver at the bottom, then IP and TCP, and eventually reaching the application specific protocol, such as Telnet, FTP and SMTP on the top. Clearly, the bottommost of these layers is somewhere within the kernel, since the physical device driver for the net is almost inevitably located there. Equally clearly, the top layers of this package, which provide the user his ability to perform the remote login function or to send mail, are not entirely contained within the kernel. Thus, the question is not whether the protocol family shall be inside or outside the kernel, but how it shall be sliced in two between that part inside and that part outside.

Since protocols come nicely layered, an obvious proposal is that one of the layer interfaces should be the point at which the inside and outside components are sliced apart. Most systems have been implemented in this way, and many have been made to work quite effectively. One obvious place to slice is at the upper interface of TCP. Since TCP provides a bidirectional byte stream, which is somewhat similar to the I/O facility provided by most operating systems, it is possible to make the interface to TCP almost mimic the interface to other existing devices. Except in the matter of opening a connection, and dealing with peculiar failures, the software using TCP need not know that it is a network connection, rather than a local I/O stream that is providing the communications function. This approach does put TCP inside the kernel, which raises all the problems addressed above. It also raises the problem that the interface to the IP layer can, if the programmer is not careful, become excessively buried inside the kernel. It must be remembered that things other than TCP are expected to run on top of IP. The IP interface must be made accessible, even if TCP sits on top of it inside the kernel.

Another obvious place to slice is above Telnet. The advantage of slicing above Telnet is that it solves the problem of having remote login channels emulate local teletype channels. The disadvantage of putting Telnet into the kernel is that the amount of code which has now been included there is getting remarkably large. In some early implementations, the size of the network package, when one includes protocols at the level of Telnet, rivals the size of the rest of the supervisor. This leads to vague feelings that all is not right.

Any attempt to slice through a lower layer boundary, for example between internet and TCP, reveals one fundamental problem. The TCP layer, as well as the IP layer, performs a demultiplexing function on incoming datagrams. Until the TCP header has been examined, it is not possible to know for which user the packet is ultimately destined. Therefore, if TCP, as a whole, is moved outside the kernel, it is necessary to create one separate process called the TCP process, which performs the TCP multiplexing function, and probably all of the rest of TCP processing as well. This means that incoming data destined for a user process involves not just a scheduling of the user process, but scheduling the TCP process first.

This suggests an alternative structuring strategy which slices through the protocols, not along an established layer boundary, but along a functional boundary having to do with demultiplexing. In this approach, certain parts of IP and certain parts of TCP are placed in the kernel. The amount of code placed there is sufficient so that when an incoming datagram arrives, it is possible to know for which process that datagram is ultimately destined. The datagram is then routed directly to the final process, where additional IP and TCP processing is performed on it. This removes from the kernel any requirement for timer based actions, since they can be done by the process provided by the user. This structure has the additional advantage of reducing the amount of code required in the kernel, so that

it is suitable for systems where kernel space is at a premium. The RFC 814, titled "Names, Addresses, Ports, and Routes," discusses this rather orthogonal slicing strategy in more detail.

A related discussion of protocol layering and multiplexing can be found in Cohen and Postel [1].

Breaking Down the Barriers

In fact, the implementor should be sensitive to the possibility of even more peculiar slicing strategies in dividing up the various protocol layers between the kernel and the one or more user processes. The result of the strategy proposed above was that part of TCP should execute in the process of the user. In other words, instead of having one TCP process for the system, there is one TCP process per connection. Given this architecture, it is not longer necessary to imagine that all of the TCPs are identical. One TCP could be optimized for high throughput applications, such as file transfer. Another TCP could be optimized for small low delay applications such as Telnet. In fact, it would be possible to produce a TCP which was somewhat integrated with the Telnet or FTP on top of it. Such an integration is extremely important, for it can lead to a kind of efficiency which more traditional structures are incapable of producing. Earlier, this paper pointed out that one of the important rules to achieving efficiency was to send the minimum number of packets for a given amount of data. The idea of protocol layering interacts very strongly (and poorly) with this goal, because independent layers have independent ideas about when packets should be sent, and unless these layers can somehow be brought into cooperation, additional packets will flow. The best example of this is the operation of server telnet in a character at a time remote echo mode on top of TCP. When a packet containing a character arrives at a server host, each layer has a different response to that packet. TCP has an obligation to acknowledge the packet. Either server telnet or the application layer above has an obligation to echo the character received in the packet. If the character is a Telnet control sequence, then Telnet has additional actions which it must perform in response to the packet. The result of this, in most implementations, is that several packets are sent back in response to the one arriving packet. Combining all of these return messages into one packet is important for several reasons. First, of course, it reduces the number of packets being sent over the net, which directly reduces the charges incurred for many common carrier tariff structures. Second, it reduces the number of scheduling actions which will occur inside both hosts, which, as was discussed above, is extremely important in improving throughput.

The way to achieve this goal of packet sharing is to break down the barrier between the layers of the protocols, in a very restrained and careful manner, so that a limited amount of information can leak across the barrier to enable one layer to optimize its behavior with respect to the desires of the layers above and below it. For example, it would represent an improvement if TCP, when it received a packet, could ask the layer above whether or not it would be worth pausing for a few milliseconds before sending an acknowledgement in order to see if the upper layer would have any outgoing data to send. Dallying before sending the acknowledgement produces precisely the right sort of optimization if the client of TCP is server Telnet. However, dallying before sending an acknowledgement is absolutely unacceptable if TCP is being used for file transfer, for in file transfer there is almost never data flowing in the reverse direction, and the delay in sending the acknowledgement probably translates directly into a delay in obtaining the next packets. Thus, TCP must know a little about the layers above it to adjust its performance as needed.

It would be possible to imagine a general purpose TCP which was equipped with all sorts of special mechanisms by which it would query the layer above and modify its behavior accordingly. In the structures suggested above, in which there is not one but several TCPs, the TCP can simply be modified so that it produces the correct behavior as a matter of course. This structure has the disadvantage that there will be several implementations of TCP existing on a single machine, which can mean more maintenance headaches if a problem is found where TCP needs to be changed. However, it is probably the case that each of the TCPs will be substantially simpler than the general purpose TCP which would

otherwise have been built. There are some experimental projects currently under way which suggest that this approach may make designing of a TCP, or almost any other layer, substantially easier, so that the total effort involved in bringing up a complete package is actually less if this approach is followed. This approach is by no means generally accepted, but deserves some consideration.

The general conclusion to be drawn from this sort of consideration is that a layer boundary has both a benefit and a penalty. A visible layer boundary, with a well specified interface, provides a form of isolation between two layers which allows one to be changed with the confidence that the other one will not stop working as a result. However, a firm layer boundary almost inevitably leads to inefficient operation. This can easily be seen by analogy with other aspects of operating systems. Consider, for example, file systems. A typical operating system provides a file system, which is a highly abstracted representation of a disk. The interface is highly formalized, and presumed to be highly stable. This makes it very easy for naive users to have access to disks without having to write a great deal of software. The existence of a file system is clearly beneficial. On the other hand, it is clear that the restricted interface to a file system almost inevitably leads to inefficiency. If the interface is organized as a sequential read and write of bytes, then there will be people who wish to do high throughput transfers who cannot achieve their goal. If the interface is a virtual memory interface, then other users will regret the necessity of building a byte stream interface on top of the memory mapped file. The most objectionable inefficiency results when a highly sophisticated package, such as a data base management package, must be built on top of an existing operating system. Almost inevitably, the implementors of the database system attempt to reject the file system and obtain direct access to the disks. They have sacrificed modularity for efficiency.

The same conflict appears in networking, in a rather extreme form. The concept of a protocol is still unknown and frightening to most naive programmers. The idea that they might have to implement a protocol, or even part of a protocol, as part of some application package, is a dreadful thought. And thus there is great pressure to hide the function of the net behind a very hard barrier. On the other hand, the kind of inefficiency which results from this is a particularly undesirable sort of inefficiency, for it shows up, among other things, in increasing the cost of the communications resource used up to achieve the application goal. In cases where one must pay for one's communications costs, they usually turn out to be the dominant cost within the system. Thus, doing an excessively good job of packaging up the protocols in an inflexible manner has a direct impact on increasing the cost of the critical resource within the system. This is a dilemma which will probably only be solved when programmers become somewhat less alarmed about protocols, so that they are willing to weave a certain amount of protocol structure into their application program, much as application programs today weave parts of database management systems into the structure of their application program.

An extreme example of putting the protocol package behind a firm layer boundary occurs when the protocol package is relegated to a front- end processor. In this case the interface to the protocol is some other protocol. It is difficult to imagine how to build close cooperation between layers when they are that far separated. Realistically, one of the prices which must be associated with an implementation so physically modularized is that the performance will suffer as a result. Of course, a separate processor for protocols could be very closely integrated into the mainframe architecture, with interprocessor co-ordination signals, shared memory, and similar features. Such a physical modularity might work very well, but there is little documented experience with this closely coupled architecture for protocol support.

RFC-817 Modularity and Efficiency in Protocol Implementation

Efficiency of Protocol Processing

To this point, this document has considered how a protocol package should be broken into modules, and how those modules should be distributed between free standing machines, the operating system kernel, and one or more user processes. It is now time to consider the other half of the efficiency question, which is what can be done to speed the execution of those programs that actually implement the protocols. We will make some specific observations about TCP and IP, and then conclude with a few generalities.

IP is a simple protocol, especially with respect to the processing of normal packets, so it should be easy to get it to perform efficiently. The only area of any complexity related to actual packet processing has to do with fragmentation and reassembly. The reader is referred to RFC 815, titled "IP Datagram Reassembly Algorithms", for specific consideration of this point.

Most costs in the IP layer come from table look up functions, as opposed to packet processing functions. An outgoing packet requires two translation functions to be performed. The internet address must be translated to a target gateway, and a gateway address must be translated to a local network number (if the host is attached to more than one network). It is easy to build a simple implementation of these table look up functions that in fact performs very poorly. The programmer should keep in mind that there may be as many as a thousand network numbers in a typical configuration. Linear searching of a thousand entry table on every packet is extremely unsuitable. In fact, it may be worth asking TCP to cache a hint for each connection, which can be handed down to IP each time a packet is sent, to try to avoid the overhead of a table look up.

TCP is a more complex protocol, and presents many more opportunities for getting things wrong. There is one area which is generally accepted as causing noticeable and substantial overhead as part of TCP processing. This is computation of the checksum. It would be nice if this cost could be avoided somehow, but the idea of an end- to-end checksum is absolutely central to the functioning of TCP. No host implementor should think of omitting the validation of a checksum on incoming data.

Various clever tricks have been used to try to minimize the cost of computing the checksum. If it is possible to add additional microcoded instructions to the machine, a checksum instruction is the most obvious candidate. Since computing the checksum involves picking up every byte of the segment and examining it, it is possible to combine the operation of computing the checksum with the operation of copying the segment from one location to another. Since a number of data copies are probably already required as part of the processing structure, this kind of sharing might conceivably pay off if it didn't cause too much trouble to the modularity of the program. Finally, computation of the checksum seems to be one place where careful attention to the details of the algorithm used can make a drastic difference in the throughput of the program. The Multics system provides one of the best case studies of this, since Multics is about as poorly organized to perform this function as any machine implementing TCP. Multics is a 36-bit word machine, with four 9-bit bytes per word. The eight-bit bytes of a TCP segment are laid down packed in memory, ignoring word boundaries. This means that when it is necessary to pick up the data as a set of 16-bit units for the purpose of adding them to compute checksums, horrible masking and shifting is required for each 16-bit value. An early version of a program using this strategy required 6 milliseconds to checksum a 576-byte segment. Obviously, at this point, checksum computation was becoming the central bottleneck to throughput. A more careful recoding of this algorithm reduced the checksum processing time to less than one millisecond. The strategy used was extremely dirty. It involved adding up carefully selected words of the area in which the data lay, knowing that for those particular words, the

16-bit values were properly aligned inside the words. Only after the addition had been done were the various sums shifted, and finally added to produce the eventual checksum. This kind of highly specialized programming is probably not acceptable if used everywhere within an operating system. It is clearly appropriate for one highly localized function which can be clearly identified as an extreme performance bottleneck.

Another area of TCP processing which may cause performance problems is the overhead of examining all of the possible flags and options which occur in each incoming packet. One paper, by Bunch and Day [2], asserts that the overhead of packet header processing is actually an important limiting factor in throughput computation. Not all measurement experiments have tended to support this result. To whatever extent it is true, however, there is an obvious strategy which the implementor ought to use in designing his program. He should build his program to optimize the expected case. It is easy, especially when first designing a program, to pay equal attention to all of the possible outcomes of every test. In practice, however, few of these will ever happen. A TCP should be built on the assumption that the next packet to arrive will have absolutely nothing special about it, and will be the next one expected in the sequence space. One or two tests are sufficient to determine that the expected set of control flags are on. (The ACK flag should be on; the Push flag may or may not be on. No other flags should be on.) One test is sufficient to determine that the sequence number of the incoming packet is one greater than the last sequence number received. In almost every case, that will be the actual result. Again, using the Multics system as an example, failure to optimize the case of receiving the expected sequence number had a detectable effect on the performance of the system. The particular problem arose when a number of packets arrived at once. TCP attempted to process all of these packets before awaking the user. As a result, by the time the last packet arrived, there was a threaded list of packets which had several items on it. When a new packet arrived, the list was searched to find the location into which the packet should be inserted. Obviously, the list should be searched from highest sequence number to lowest sequence number, because one is expecting to receive a packet which comes after those already received. By mistake, the list was searched from front to back, starting with the packets with the lowest sequence number. The amount of time spent searching this list backwards was easily detectable in the metering measurements.

Other data structures can be organized to optimize the action which is normally taken on them. For example, the retransmission queue is very seldom actually used for retransmission, so it should not be organized to optimize that action. In fact, it should be organized to optimized the discarding of things from it when the acknowledgement arrives. In many cases, the easiest way to do this is not to save the packet at all, but to reconstruct it only if it needs to be retransmitted, starting from the data as it was originally buffered by the user.

There is another generality, at least as important as optimizing the common case, which is to avoid copying data any more times than necessary. One more result from the Multics TCP may prove enlightening here. Multics takes between two and three milliseconds within the TCP layer to process an incoming packet, depending on its size. For a 576- byte packet, the three milliseconds is used up approximately as follows. One millisecond is used computing the checksum. Six hundred microseconds is spent copying the data. (The data is copied twice, at .3 milliseconds a copy.) One of those copy operations could correctly be included as part of the checksum cost, since it is done to get the data on a known word boundary to optimize the checksum algorithm. However, the copy also performs another necessary transfer at the same time. Header processing and packet resequencing takes .7 milliseconds. The rest of the time is used in miscellaneous processing, such as removing packets from the retransmission queue which are acknowledged by this packet. Data copying is the second most expensive single operation after data checksuming. Some implementations, often because of an excessively layered modularity, end up copying the data around a great deal. Other implementations end up copying the data because there is no shared memory between processes, and the data must be moved from process to

process via a kernel operation. Unless the amount of this activity is kept strictly under control, it will quickly become the major performance bottleneck.

Conclusions

This document has addressed two aspects of obtaining performance from a protocol implementation, the way in which the protocol is layered and integrated into the operating system, and the way in which the detailed handling of the packet is optimized. It would be nice if one or the other of these costs would completely dominate, so that all of one's attention could be concentrated there. Regrettably, this is not so. Depending on the particular sort of traffic one is getting, for example, whether Telnet one-byte packets or file transfer maximum size packets at maximum speed, one can expect to see one or the other cost being the major bottleneck to throughput. Most implementors who have studied their programs in an attempt to find out where the time was going have reached the unsatisfactory conclusion that it is going equally to all parts of their program. With the possible exception of checksum processing, very few people have ever found that their performance problems were due to a single, horrible bottleneck which they could fix by a single stroke of inventive programming. Rather, the performance was something which was improved by painstaking tuning of the entire program.

Most discussions of protocols begin by introducing the concept of layering, which tends to suggest that layering is a fundamentally wonderful idea which should be a part of every consideration of protocols. In fact, layering is a mixed blessing. Clearly, a layer interface is necessary whenever more than one client of a particular layer is to be allowed to use that same layer. But an interface, precisely because it is fixed, inevitably leads to a lack of complete understanding as to what one layer wishes to obtain from another. This has to lead to inefficiency. Furthermore, layering is a potential snare in that one is tempted to think that a layer boundary, which was an artifact of the specification procedure, is in fact the proper boundary to use in modularizing the implementation. Again, in certain cases, an architected layer must correspond to an implemented layer, precisely so that several clients can have access to that layer in a reasonably straightforward manner. In other cases, cunning rearrangement of the implemented module boundaries to match with various functions, such as the demultiplexing of incoming packets, or the sending of asynchronous outgoing packets, can lead to unexpected performance improvements compared to more traditional implementation strategies. Finally, good performance is something which is difficult to retrofit onto an existing program. Since performance is influenced, not just by the fine detail, but by the gross structure, it is sometimes the case that in order to obtain a substantial performance improvement, it is necessary to completely redo the program from the bottom up. This is a great disappointment to programmers, especially those doing a protocol implementation for the first time. Programmers who are somewhat inexperienced and unfamiliar with protocols are sufficiently concerned with getting their program logically correct that they do not have the capacity to think at the same time about the performance of the structure they are building. Only after they have achieved a logically correct program do they discover that they have done so in a way which has precluded real performance. Clearly, it is more difficult to design a program thinking from the start about both logical correctness and performance. With time, as implementors as a group learn more about the appropriate structures to use for building protocols, it will be possible to proceed with an implementation project having more confidence that the structure is rational, that the program will work, and that the program will work well. Those of us now implementing protocols have the privilege of being on the forefront of this learning process. It should be no surprise that our programs sometimes suffer from the uncertainty we bring to bear on them.

Citations

[1] Cohen and Postel, "On Protocol Multiplexing", Sixth Data Communications Symposium, ACM/IEEE, November 1979.

[2] Bunch and Day, "Control Structure Overhead in TCP", Trends and Applications: Computer Networking, NBS Symposium, May 1980.

RFC-821 SIMPLE MAIL TRANSFER PROTOCOL

Jonathan B. Postel
August 1982
Information Sciences Institute
University of Southern California

RFC-821--SIMPLE MAIL TRANSFER PROTOCOL

INTRODUCTION

The objective of Simple Mail Transfer Protocol (SMTP) is to transfer mail reliably and efficiently.

SMTP is independent of the particular transmission subsystem and requires only a reliable ordered data stream channel. Appendices A, B, C, and D describe the use of SMTP with various transport services. A Glossary provides the definitions of terms as used in this document.

An important feature of SMTP is its capability to relay mail across transport service environments. A transport service provides an interprocess communication environment (IPCE). An IPCE may cover one network, several networks, or a subset of a network. It is important to realize that transport systems (or IPCEs) are not one-to-one with networks. A process can communicate directly with another process through any mutually known IPCE. Mail is an application or use of interprocess communication. Mail can be communicated between processes in different IPCEs by relaying through a process connected to two (or more) IPCEs. More specifically, mail can be relayed between hosts on different transport systems by a host on both transport systems.

RFC-821--SIMPLE MAIL TRANSFER PROTOCOL

The SMTP Model

Mail is sent by a series of request/response transactions between a client, the "sender-SMTP," and a server, the "receiver-SMTP". These transactions pass (1) the message proper, which is composed of header and body, and (2) SMTP source and destination addresses, referred to as the "envelope".

The SMTP programs are analogous to Message Transfer Agents (MTAs) of X.400. There will be another level of protocol software, closer to the end user, that is responsible for composing and analyzing RFC-822 message headers; this component is known as the "User Agent" in X.400, and we use that term in this document. There is a clear logical distinction between the User Agent and the SMTP implementation, since they operate on different levels of protocol. Note, however, that this distinction is may not be exactly reflected the structure of typical implementations of Internet mail. Often there is a program known as the "mailer" that implements SMTP and also some of the User Agent functions; the rest of the User Agent functions are included in a user interface used for entering and reading mail.

The SMTP envelope is constructed at the originating site, typically by the User Agent when the message is first queued for the Sender-SMTP program. The envelope addresses may be derived from information in the message header, supplied by the user interface (e.g., to implement a bcc: request), or derived from local configuration information (e.g., expansion of a mailing list). The SMTP envelope cannot in general be re-derived from the header at a later stage in message delivery, so the envelope is transmitted separately from the message itself using the MAIL and RCPT commands of SMTP.

With the advent of the domain system and of mail routing using mail-exchange (MX) resource records, implementors should now think of delivering mail to a user at a domain, which may or may not be a particular host. This does not change the fact that SMTP is a host-to-host mail exchange protocol.

The SMTP design is based on the following model of communication: as the result of a user mail request, the sender-SMTP establishes a two-way transmission channel to a receiver-SMTP. The receiver-SMTP may be either the ultimate destination or an intermediate. SMTP commands are generated by the sender-SMTP and sent to the receiver-SMTP. SMTP replies are sent from the receiver-SMTP to the sender-SMTP in response to the commands.

Once the transmission channel is established, the SMTP-sender sends a MAIL command indicating the sender of the mail. If the SMTP-receiver can accept mail it responds with an OK reply. The SMTP-sender then sends a RCPT command identifying a recipient of the mail. If the SMTP-receiver can accept mail for that recipient it responds with an OK reply; if not, it responds with a reply rejecting that recipient (but not the whole mail transaction). The SMTP-sender and SMTP-receiver may negotiate several recipients. When the recipients have been negotiated the SMTP-sender sends the mail data, terminating with a special sequence. If the SMTP-receiver successfully processes the mail data it responds with an OK reply. The dialog is purposely lock-step, one-at-a-time.

Model for SMTP Use

The SMTP provides mechanisms for the transmission of mail; directly from the sending user's host to the receiving user's host when the two host are connected to the same transport service, or via one or more relay SMTP-servers when the source and destination hosts are not connected to the same transport service.

To be able to provide the relay capability the SMTP-server must be supplied with the name of the ultimate destination host as well as the destination mailbox name.

The argument to the MAIL command is a reverse-path, which specifies who the mail is from. The argument to the RCPT command is a forward-path, which specifies who the mail is to. The forward-path s a source route, while the reverse-path is a return route (which may be used to return a message to the sender when an error occurs with a relayed message).

When the same message is sent to multiple recipients the SMTP encourages the transmission of only one copy of the data for all the recipients at the same destination host.

The mail commands and replies have a rigid syntax. Replies also have a numeric code. In the following, examples appear which use actual commands and replies. The complete lists of commands and replies appears in the section on specifications.

Commands and replies are not case sensitive. That is, a command or reply word may be upper case, lower case, or any mixture of upper and lower case. Note that this is not true of mailbox user names. For some hosts the user name is case sensitive, and SMTP implementations must take case to preserve the case of user names as they appear in mailbox arguments. Host names are not case sensitive.

Commands and replies are composed of characters from the ASCII character set. When the transport service provides an 8-bit byte (octet) transmission channel, each 7-bit character is transmitted right justified in an octet with the high order bit cleared to zero.

When specifying the general form of a command or reply, an argument (or special symbol) will be denoted by a meta-linguistic variable (or constant), for example, "<string>" or "<reverse-path>". Here the angle brackets indicate these are meta-linguistic variables. However, some arguments use the angle brackets literally. For example, an actual reverse-path is enclosed in angle brackets, i.e., "<John.Smith@USC-ISI.ARPA>" is an instance of <reverse-path> (the angle brackets are actually transmitted in the command or reply). Scenarios are presented in Appendix F.

RFC-821--SIMPLE MAIL TRANSFER PROTOCOL

The SMTP Procedure

RFC-821--SIMPLE MAIL TRANSFER PROTOCOL

MAIL

There are three steps to SMTP mail transactions. The transaction is started with a MAIL command which gives the sender identification. A series of one or more RCPT commands follows giving the receiver information. Then a DATA command gives the mail data. And finally, the end of mail data indicator confirms the transaction.

Canonicalization
The domain names that a Sender-SMTP sends in MAIL and RCPT commands **must** have been "canonicalized," i.e., they must be fully-qualified principal names or domain literals, not nicknames or domain abbreviations. A canonicalized name either identifies a host directly or is an MX name; it cannot be a CNAME.

The first step in the procedure is the MAIL command. The <reverse-path> contains the source mailbox.

```
MAIL <SP> FROM:<reverse-path> <CRLF>
```

This command tells the SMTP-receiver that a new mail transaction is starting and to reset all its state tables and buffers, including any recipients or mail data. It gives the reverse-path which can be used to report errors. If accepted, the receiver-SMTP returns a 250 OK reply.

The <`reverse-path`> can contain more than just a mailbox. The <`reverse-path`> is a reverse source routing list of hosts and source mailbox. The first host in the <`reverse-path`> should be the host sending this command.

The second step in the procedure is the RCPT command.

```
RCPT <SP> TO:<forward-path> <CRLF>
```

This command gives a forward-path identifying one recipient. If accepted, the receiver-SMTP returns a 250 OK reply, and stores the forward-path. If the recipient is unknown the receiver-SMTP returns a 550 Failure reply. This second step of the procedure can be repeated any number of times.

The <`forward-path`> can contain more than just a mailbox. The <`forward-path`> is a source routing list of hosts and the destination mailbox. The first host in the <`forward-path`> should be the host receiving this command.

The third step in the procedure is the DATA command.

```
DATA <CRLF>
```

If accepted, the receiver-SMTP returns a 354 Intermediate reply and considers all succeeding lines to be the message text. When the end of text is received and stored the SMTP-receiver sends a 250 OK reply.

Since the mail data is sent on the transmission channel the end of the mail data must be indicated so that the command and reply dialog can be resumed. SMTP indicates the end of the mail data by sending a line containing only a period. A transparency procedure is used to prevent this from interfering with the user's text.

Please note that the mail data includes the memo header items such as Date, Subject, To, Cc, From (RFC-822).

The end of mail data indicator also confirms the mail transaction and tells the receiver-SMTP to now process the stored recipients and mail data. If accepted, the receiver-SMTP returns a 250 OK reply. The DATA command should fail only if the mail transaction was incomplete (for example, no recipients), or if resources are not available.

The above procedure is an example of a mail transaction. These commands must be used only in the order discussed above. The example below illustrates the use of these commands in a mail transaction.

Example of the SMTP Procedure

This SMTP example shows mail sent by Smith at host Alpha.ARPA, to Jones, Green, and Brown at host Beta.ARPA. Here we assume that host Alpha contacts host Beta directly.

```
S: MAIL FROM:<Smith@Alpha.ARPA>
R: 250 OK

S: RCPT TO:<Jones@Beta.ARPA>
R: 250 OK

S: RCPT TO:<Green@Beta.ARPA>
R: 550 No such user here

S: RCPT TO:<Brown@Beta.ARPA>
R: 250 OK

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: Blah blah blah...
S: ...etc. etc. etc.
S: <CRLF>.<CRLF>
R: 250 OK
```

The mail has now been accepted for Jones and Brown. Green did not have a mailbox at host Beta.

Forwarding

There are some cases where the destination information in the <forward-path> is incorrect, but the receiver-SMTP knows the correct destination. In such cases, one of the following replies should be used to allow the sender to contact the correct destination.

```
251 User not local; will forward to <forward-path>
```

This reply indicates that the receiver-SMTP knows the user's mailbox is on another host and indicates the correct forward-path to use in the future. Note that either the host or user or both may be different. The receiver takes responsibility for delivering the message.

```
551 User not local; please try <forward-path>
```

This reply indicates that the receiver-SMTP knows the user's mailbox is on another host and indicates the correct forward-path to use. Note that either the host or user or both may be different. The receiver refuses to accept mail for this user, and the sender must either redirect the mail according to the information provided or return an error response to the originating user.

Example of Forwarding

Either

```
S: RCPT TO:<Postel@USC-ISI.ARPA>
R: 251 User not local; will forward to <Postel@USC-ISIF.ARPA>
```

Or

```
S: RCPT TO:<Paul@USC-ISIB.ARPA>
R: 551 User not local; please try <Mockapetris@USC-ISIF.ARPA>
```

Verifying And Expanding

SMTP provides as additional features, commands to verify a user name or expand a mailing list. This is done with the VRFY and EXPN commands, which have character string arguments. For the VRFY command, the string is a user name, and the response may include the full name of the user and must include the mailbox of the user. For the EXPN command, the string identifies a mailing list, and the multiline response may include the full name of the users and must give the mailboxes on the mailing list.

"User name" is a fuzzy term and used purposely. When a host implements the VRFY or EXPN commands then at least local mailboxes must be recognized as "user names". If a host chooses to recognize other strings as "user names" that is allowed.

A receiver-SMTP **must** implement VRFY and **should** implement EXPN. However, there **may** be configuration information to disableVRFY and EXPN in a particular installation; this might even allow EXPN to be disabled for selected lists.

> **Discussion**
> SMTP users and administrators make regular use of these commands for diagnosing mail delivery problems. With the increasing use of multi-level mailing list expansion (sometimes more than two levels), EXPN has been increasingly important for diagnosing inadvertent mail loops. On the other hand, some feel that EXPN represents a significant privacy, and perhaps even a security, exposure.

In some hosts the distinction between a mailing list and an alias for a single mailbox is a bit fuzzy, since a common data structure may hold both types of entries, and it is possible to have mailing lists of one mailbox. If a request is made to verify a mailing list a positive response can be given if on receipt of a message so addressed it will be delivered to everyone on the list, otherwise an error should be reported (e.g., "550 That is a mailing list, not a user"). If a request is made to expand a user name a positive response can be formed by returning a list containing one name, or an error can be reported (e.g., "550 That is a user name, not a mailing list").

In the case of a multiline reply (normal for EXPN) exactly one mailbox is to be specified on each line of the reply. In the case of an ambiguous request, for example, "VRFY Smith", where there are two Smith's the response must be "553 User ambiguous".

The case of verifying a user name is straightforward as shown in the example below.

Example of Verifying a User Name

Either

```
S: VRFY Smith
R: 250 Fred Smith <Smith@USC-ISIF.ARPA>
```

Or

```
S: VRFY Smith
R: 251 User not local; will forward to <Smith@USC-ISIQ.ARPA>
```

Or

```
S: VRFY Jones
R: 550 String does not match anything.
```

Or

```
S: VRFY Jones
R: 551 User not local; please try <Jones@USC-ISIQ.ARPA>
```

Or

```
S: VRFY Gourzenkyinplatz
R: 553 User ambiguous.
```

The case of expanding a mailbox list requires a multiline reply as shown in the following example.

Example of Expanding a Mailing List

Either

```
S: EXPN Example-People
R: 250-Jon Postel <Postel@USC-ISIF.ARPA>
R: 250-Fred Fonebone <Fonebone@USC-ISIQ.ARPA>
R: 250-Sam Q. Smith <SQSmith@USC-ISIQ.ARPA>
R: 250-Quincy Smith <@USC-ISIF.ARPA:Q-Smith@ISI-VAXA.ARPA>
R: 250-<joe@foo-unix.ARPA>
R: 250 <xyz@bar-unix.ARPA>
```

Or

```
S: EXPN Executive-Washroom-List
R: 550 Access Denied to You.
```

The character string arguments of the VRFY and EXPN commands cannot be further restricted due to the variety of implementations of the user name and mailbox list concepts. On some systems it may be appropriate for the argument of the EXPN command to be a file name for a file containing a mailing list, but again there is a variety of file naming conventions in the Internet.

The EXPN command is not included in the minimum implementation , and is not required to work across relays when it is implemented.

Sending and Mailing

The main purpose of SMTP is to deliver messages to user's mailboxes. A very similar service provided by some hosts is to deliver messages to user's terminals (provided the user is active on the host). The delivery to the user's mailbox is called "mailing", the delivery to the user's terminal is called "sending". Because in many hosts the implementation of sending is nearly identical to the implementation of mailing these two functions are combined in SMTP. However the sending commands are not included in the required minimum implementation. Users should have the ability to control the writing of messages on their terminals. Most hosts permit the users to accept or refuse such messages.

> **Discussion**
> It has been suggested that the use of mail relaying through an MX record is inconsistent with the intent of SEND to deliver a message immediately and directly to a user's terminal. However, an SMTP receiver that is unable to write directly to the user terminal can return a "251 User Not Local" reply to the RCPT following a SEND, to inform the originator of possibly deferred delivery.

The following three command are defined to support the sending options. These are used in the mail transaction instead of the MAIL command and inform the receiver-SMTP of the special semantics of this transaction:

```
SEND <SP> FROM:<reverse-path> <CRLF>
```

The SEND command requires that the mail data be delivered to the user's terminal. If the user is not active (or not accepting terminal messages) on the host a 450 reply may returned to a RCPT command. The mail transaction is successful if the message is delivered the terminal.

SOML <S> FROM:<reverse-path> <CRLF>

The Send Or MaiL command requires that the mail data be delivered to the user's terminal if the user is active (and accepting terminal messages) on the host. If the user is not active (or not accepting terminal messages) then the mail data is entered into the user's mailbox. The mail transaction is successful if the message is delivered either to the terminal or the mailbox.

SAML <SP> FROM:<reverse-path> <CRLF>

The Send And MaiL command requires that the mail data be delivered to the user's terminal if the user is active (and accepting terminal messages) on the host. In any case the mail data is entered into the user's mailbox. The mail transaction is successful if the message is delivered the mailbox.

The same reply codes that are used for the MAIL commands are used for these commands.

Opening and Closing

At the time the transmission channel is opened there is an exchange to ensure that the hosts are communicating with the hosts they think they are.

The following two commands are used in transmission channel opening and closing:

```
HELO <SP> <domain> <CRLF>

QUIT <CRLF>
```

In the HELO command the host sending the command identifies itself; the command may be interpreted as saying "Hello, I am <domain>".

Example of Connection Opening

```
R: 220 BBN-UNIX.ARPA Simple Mail Transfer Service Ready
S: HELO USC-ISIF.ARPA
R: 250 BBN-UNIX.ARPA
```

Example of Connection Closing

```
S: QUIT
R: 221 BBN-UNIX.ARPA Service closing transmission channel
```

Relaying

We distinguish three types of mail (store-and-) forwarding:

(1) A simple forwarder or "mail exchanger" forwards a message using private knowledge about the recipient; see section on Forwarding.

(2) An SMTP mail "relay" forwards a message within an SMTP mail environment as the result of an explicit source route as defined in below). The SMTP relay function uses the "@...:" form of source route from RFC- 822 (see Explicit Path Specification).

(3) A mail "gateway" passes a message between different environments.

An Internet host that is forwarding a message but is not a gateway to a different mail environment (i.e., it falls under (1) or (2)) **should not** alter any existing header fields, although the host will add an appropriate Received: line as required in DATA Command.

A Sender-SMTP **should not** send a RCPT TO: command containing an explicit source route using the "@...:" address form. Thus, the relay function defined in below should not be used.

Discussion
The intent is to discourage all source routing and to abolish explicit source routing for mail delivery within the Internet environment. Source-routing is unnecessary; the simple target address "user@domain" should always suffice. This is the result of an explicit architectural decision to use universalnaming rather than source routing for mail. Thus, SMTP provides end-to-end connectivity, and the DNS provides globally-unique, location-independent names. MXrecords handle the major case where source routing might otherwise be needed.

A receiver-SMTP **must** accept the explicit source route syntax in the envelope, but it **may** implement the relay function as defined below. If it does not implement the relay function, it **should** attempt to deliver the message directly to the host to the right of the right-most "@" sign.

Discussion
For example, suppose a host that does not implement the relay function receives a message with the SMTP command: "RCPT TO:<@ALPHA,@BETA:joe@GAMMA>", where ALPHA, BETA, and GAMMA represent domain names. Rather than immediately refusing the message with a 550 error reply as suggested in the example below, the host should try to forward the message to GAMMA directly, using: "RCPT TO:<joe@GAMMA>". Since this host does not support relaying, it is not required to update the reverse path.

Some have suggested that source routing may be needed occasionally for manually routing mail around failures; however, the reality and importance of this need is controversial. The use of explicit SMTP mail relaying for this purpose is discouraged, and in fact it may not be successful, as many host systems do not support it. Some have used the "%-hack" (see Balancing

Local Part and Domain) for this purpose.

The forward-path may be a source route of the form "@ONE,@TWO:JOE@THREE", where ONE, TWO, and THREE are hosts. This form is used to emphasize the distinction between an address and a route. The mailbox is an absolute address, and the route is information about how to get there. The two concepts should not be confused.

Conceptually the elements of the forward-path are moved to the reverse-path as the message is relayed from one server-SMTP to another. The reverse-path is a reverse source route, (i.e., a source route from the current location of the message to the originator of the message). When a server-SMTP deletes its identifier from the forward-path and inserts it into the reverse-path, it must use the name it is known by in the environment it is sending into, not the environment the mail came from, in case the server-SMTP is known by different names in different environments.

If when the message arrives at an SMTP the first element of the forward-path is not the identifier of that SMTP the element is not deleted from the forward-path and is used to determine the next SMTP to send the message to. In any case, the SMTP adds its own identifier to the reverse-path.

Using source routing the receiver-SMTP receives mail to be relayed to another server-SMTP The receiver-SMTP may accept or reject the task of relaying the mail in the same way it accepts or rejects mail for a local user. The receiver-SMTP transforms the command arguments by moving its own identifier from the forward-path to the beginning of the reverse-path. The receiver-SMTP then becomes a sender-SMTP, establishes a transmission channel to the next SMTP in the forward-path, and sends it the mail.

The first host in the reverse-path should be the host sending the SMTP commands, and the first host in the forward-path should be the host receiving the SMTP commands.

Notice that the forward-path and reverse-path appear in the SMTP commands and replies, but not necessarily in the message. That is, there is no need for these paths and especially this syntax to appear in the "To:" , "From:", "CC:", etc. fields of the message header.

If a server-SMTP has accepted the task of relaying the mail and later finds that the forward-path is incorrect or that the mail cannot be delivered for whatever reason, then it must construct an "undeliverable mail" notification message and send it to the originator of the undeliverable mail (as indicated by the reverse-path).

This notification message must be from the server-SMTP at this host. Of course, server-SMTPs should not send notification messages about problems with notification messages. One way to prevent loops in error reporting is to specify a null reverse-path in the MAIL command of a notification message. When such a message is relayed it is permissible to leave the reverse-path null. A MAIL command with a null reverse-path appears as follows:

```
MAIL FROM:<>
```

An undeliverable mail notification message is shown in the example below. This notification is in response to a message originated by JOE at HOSTW and sent via HOSTX to HOSTY with instructions to relay it on to HOSTZ. What we see in the example is the transaction between HOSTY and HOSTX, which is the first step in the return of the notification message.

Example Undeliverable Mail Notification Message

```
S: MAIL FROM:<>
R: 250 ok
S: RCPT TO:<@HOSTX.ARPA:JOE@HOSTW.ARPA>
R: 250 ok
S: DATA
R: 354 send the mail data, end with .
S: Date: 23 Oct 81 11:22:33
S: From: SMTP@HOSTY.ARPA
S: To: JOE@HOSTW.ARPA
S: Subject: Mail System Problem
S:
S:   Sorry JOE, your message to SAM@HOSTZ.ARPA lost.
S:   HOSTZ.ARPA said this:
S:    "550 No Such User"
S: .
R: 250 ok
```

Domains

Domains are a recently introduced concept in the ARPA Internet mail system. The use of domains changes the address space from a flat global space of simple character string host names to a hierarchically structured rooted tree of global addresses. The host name is replaced by a domain and host designator which is a sequence of domain element strings separated by periods with the understanding that the domain elements are ordered from the most specific to the most general.

For example, "`USC-ISIF.ARPA`", "`Fred.Cambridge.UK`", and "`PC7.LCS.MIT.ARPA`" might be host-and-domain identifiers.

Whenever domain names are used in SMTP only the official names are used, the use of nicknames or aliases is not allowed.

Changing Roles

The TURN command may be used to reverse the roles of the two programs communicating over the transmission channel.

If program-A is currently the sender-SMTP and it sends the TURN command and receives an ok reply (250) then program-A becomes the receiver-SMTP.

If program-B is currently the receiver-SMTP and it receives the TURN command and sends an ok reply (250) then program-B becomes the sender-SMTP.

To refuse to change roles the receiver sends the 502 reply.

Please note that this command is optional. It would not normally be used in situations where the transmission channel is TCP. However, when the cost of establishing the transmission channel is high, this command may be quite useful. For example, this command may be useful in supporting be mail exchange using the public switched telephone system as a transmission channel, especially if some hosts poll other hosts for mail exchanges.

RFC-821--SIMPLE MAIL TRANSFER PROTOCOL

Specifications

RFC-821--SMTP Specifications

Command Semantics

The SMTP commands define the mail transfer or the mail system function requested by the user. SMTP commands are character strings terminated by <CRLF>. The command codes themselves are alphabetic characters terminated by <SP> if parameters follow and <CRLF> otherwise. The syntax of mailboxes must conform to receiver site conventions. The SMTP commands are discussed below. The SMTP replies are discussed in a separate section of SMTP Specifications.

A mail transaction involves several data objects which are communicated as arguments to different commands. The reverse-path is the argument of the MAIL command, the forward-path is the argument of the RCPT command, and the mail data is the argument of the DATA command. These arguments or data objects must be transmitted and held pending the confirmation communicated by the end of mail data indication which finalizes the transaction. The model for this is that distinct buffers are provided to hold the types of data objects, that is, there is a reverse-path buffer, a forward-path buffer, and a mail data buffer. Specific commands cause information to be appended to a specific buffer, or cause one or more buffers to be cleared.

There are restrictions on the order in which these command may be used.

The first command in a session must be the HELO command. The HELO command may be used later in a session as well. If the HELO command argument is not acceptable a 501 failure reply must be returned and the receiver-SMTP must stay in the same state.

The NOOP, HELP, EXPN, and VRFY commands can be used at any time during a session.

The MAIL, SEND, SOML, or SAML commands begin a mail transaction. Once started a mail transaction consists of one of the transaction beginning commands, one or more RCPT commands, and a DATA command, in that order. A mail transaction may be aborted by the RSET command. There may be zero or more transactions in a session.

If the transaction beginning command argument is not acceptable a 501 failure reply must be returned and the receiver-SMTP must stay in the same state. If the commands in a transaction are out of order a 503 failure reply must be returned and the receiver-SMTP must stay in the same state.

The last command in a session must be the QUIT command. The QUIT command can not be used at any other time in a session.

RFC-821--SMTP Specifications

Commands

RFC-821--SMTP Specifications: Commands

HELLO (HELO)

This command is used to identify the sender-SMTP to the receiver-SMTP. The argument field contains the host name of the sender-SMTP.

The receiver-SMTP identifies itself to the sender-SMTP in the connection greeting reply, and in the response to this command.

This command and an OK reply to it confirm that both the sender-SMTP and the receiver-SMTP are in the initial state, that is, there is no transaction in progress and all state tables and buffers are cleared.

The sender-SMTP **must** ensure that the <domain> parameter in a HELO command is a valid principal host domain name for the client host. As a result, the receiver-SMTP will not have to perform MX resolution on this name in order to validate the HELO parameter.

The HELO receiver **may** verify that the HELO parameter really corresponds to the IP address of the sender. However, the receiver **must not** refuse to accept a message, even if the sender's HELO command fails verification.

Discussion
Verifying the HELO parameter requires a domain name lookup and may therefore take considerable time. An alternative tool for tracking bogus mail sources is suggested below (see "DATA Command").

Note also that the HELO argument is still required to have valid <domain> syntax, since it will appear in a Received: line; otherwise, a 501 error is to be sent.

Implementation
When HELO parameter validation fails, a suggested procedure is to insert a note about the unknown authenticity of the sender into the message header (e.g., in the "Received:" line).

MAIL (MAIL)

This command is used to initiate a mail transaction in which the mail data is delivered to one or more mailboxes. The argument field contains a reverse-path.

The reverse-path consists of an optional list of hosts and the sender mailbox. When the list of hosts is present, it is a "reverse" source route and indicates that the mail was relayed through each host on the list (the first host in the list was the most recent relay). This list is used as a source route to return non-delivery notices to the sender. As each relay host adds itself to the beginning of the list, it must use its name as known in the IPCE to which it is relaying the mail rather than the IPCE from which the mail came (if they are different). In some types of error reporting messages (for example, undeliverable mail notifications) the reverse-path may be null (see example under Relaying).

This command clears the reverse-path buffer, the forward-path buffer, and the mail data buffer; and inserts the reverse-path information from this command into the reverse-path buffer.

RFC-821--SMTP Specifications: Commands

RECIPIENT (RCPT)

This command is used to identify an individual recipient of the mail data; multiple recipients are specified by multiple use of this command.

The forward-path consists of an optional list of hosts and a required destination mailbox. When the list of hosts is present, it is a source route and indicates that the mail must be relayed to the next host on the list. If the receiver-SMTP does not implement the relay function it may user the same reply it would for an unknown local user (550).

A host that supports a receiver-SMTP **must** support the reserved mailbox "Postmaster".

The receiver-SMTP **may** verify RCPT parameters as they arrive; however, RCPT responses **must not** be delayed beyond a reasonable time.

Therefore, a "250 OK" response to a RCPT does not necessarily imply that the delivery address(es) are valid. Errors found after message acceptance will be reported by mailing a notification message to an appropriate address (see Reliable Mail Receipt).

> **Discussion**
> The set of conditions under which a RCPT parameter can be validated immediately is an engineering design choice. Reporting destination mailbox errors to the Sender-SMTP before mail is transferred is generally desirable to save time and network bandwidth, but this advantage is lost if RCPT verification islengthy.
>
> For example, the receiver can verify immediately any simple local reference, such as a single locally- registered mailbox. On the other hand, the "reasonable time" limitation generally implies deferring verification of a mailing list until after the message has been transferred and accepted, since verifying a large mailing list can take a very long time. An implementation might or might not choose to defer validation of addresses that are non-local and therefore require a DNS lookup. If a DNS lookup is performed but a soft domain system error (e.g., timeout) occurs, validity must be assumed.

When mail is relayed, the relay host must remove itself from the beginning forward-path and put itself at the beginning of the reverse-path. When mail reaches its ultimate destination (the forward-path contains only a destination mailbox), the receiver-SMTP inserts it into the destination mailbox in accordance with its host mail conventions.

For example, mail received at relay host A with arguments

```
FROM:<USERX@HOSTY.ARPA>
TO:<@HOSTA.ARPA,@HOSTB.ARPA:USERC@HOSTD.ARPA>
```

will be relayed on to host B with arguments

```
FROM:<@HOSTA.ARPA:USERX@HOSTY.ARPA>
TO:<@HOSTB.ARPA:USERC@HOSTD.ARPA>.
```

This command causes its forward-path argument to be appended to the forward-path buffer.

RFC-821--SMTP Specifications: Commands

DATA (DATA)

The receiver treats the lines following the command as mail data from the sender. This command causes the mail data from this command to be appended to the mail data buffer. The mail data may contain any of the 128 ASCII character codes.

The mail data is terminated by a line containing only a period, that is the character sequence "<CRLF>.<CRLF>" (see the section on Transparency). This is the end of mail data indication.

The end of mail data indication requires that the receiver must now process the stored mail transaction information. This processing consumes the information in the reverse-path buffer, the forward-path buffer, and the mail data buffer, and on the completion of this command these buffers are cleared. If the processing is successful the receiver must send an OK reply. If the processing fails completely the receiver must send a failure reply.

Every receiver-SMTP **must** insert, at the beginning of the mail data, a "Received:" line, also called a time stamp line, with the followng characteristics:

- The FROM field **should** contain both (1) the name of the source host as presented in the HELO command and (2) a domain literal containing the IP address of the source, determined from the TCP connection.
- The ID field **may** contain an "@", but this is not required.
- The FOR field **may** contain a list of <path> entries when multiple RCPT commands have been given.

An Internet mail program **must not** change a Received: line that was previously added to the message header.

Discussion
Including both the source host and the IP source address in the Received: line may provide enough information for tracking illicit mail sources and eliminate a need to explicitly verify the HELO parameter.

Received: lines are primarily intended for humans tracing mail routes, primarily for diagnosis of faults.

When the receiver-SMTP makes "final delivery" of a message, then it **must** pass the MAIL FROM: address from the SMTP envelope with the message, for use if an error notification message must be sent later (see Reliable Mail Receipt). There is an analogous requirement when gatewaying from the Internet into a different mail environment.

Discussion
Note that the final reply to the DATA command depends only upon the successful transfer and storage of the message. Any problem with the destination address(es) must either (1) have been reported in an SMTP error reply to the RCPT command(s), or (2) be reported in a later error message mailed to the originator.

Implementation
The MAIL FROM: information may be passed as a parameter or in a Return-Path: line inserted at the beginning of the message.

When the receiver-SMTP makes the "final delivery" of a message it inserts at the beginning of the mail data a return path line. The return path line preserves the information in the <reverse-path> from the MAIL command. Here, final delivery means the message leaves the SMTP world. Normally, this would mean it has been delivered to the destination user, but in some cases it may be further processed and transmitted by another mail system.

It is possible for the mailbox in the return path be different from the actual sender's mailbox, for example, if error responses are to be delivered a special error handling mailbox rather than the message senders.

The preceding two paragraphs imply that the final mail data will begin with a return path line, followed by one or more time stamp lines. These lines will be followed by the mail data header and body (RFC-822). See the example below.

Special mention is needed of the response and further action required when the processing following the end of mail data indication is partially successful. This could arise if after accepting several recipients and the mail data, the receiver-SMTP finds that the mail data can be successfully delivered to some of the recipients, but it cannot be to others (for example, due to mailbox space allocation problems). In such a situation, the response to the DATA command must be an OK reply. But, the receiver-SMTP must compose and send an "undeliverable mail" notification message to the originator of the message. Either a single notification which lists all of the recipients that failed to get the message, or separate notification messages must be sent for each failed recipient (see example under Relaying). All undeliverable mail notification messages are sent using the MAIL command (even if they result from processing a SEND, SOML, or SAML command).

Example of Return Path and Received Time Stamps

Return-Path:
```
<@GHI.ARPA,@DEF.ARPA,@ABC.ARPA:JOE@ABC.ARPA>
Received: from GHI.ARPA by JKL.ARPA ; 27 Oct 81 15:27:39 PST
Received: from DEF.ARPA by GHI.ARPA ; 27 Oct 81 15:15:13 PST
Received: from ABC.ARPA by DEF.ARPA ; 27 Oct 81 15:01:59 PST
Date: 27 Oct 81 15:01:01 PST
From: JOE@ABC.ARPA
Subject: Improved Mailing System Installed To: SAM@JKL.ARPA

This is to inform you that ...
```

SEND (SEND)

This command is used to initiate a mail transaction in which the mail data is delivered to one or more terminals. The argument field contains a reverse-path. This command is successful if the message is delivered to a terminal.

The reverse-path consists of an optional list of hosts and the sender mailbox. When the list of hosts is present, it is a "reverse" source route and indicates that the mail was relayed through each host on the list (the first host in the list was the most recent relay). This list is used as a source route to return non-delivery notices to the sender. As each relay host adds itself to the beginning of the list, it must use its name as known in the IPCE to which it is relaying the mail rather than the IPCE from which the mail came (if they are different).

This command clears the reverse-path buffer, the forward-path buffer, and the mail data buffer; and inserts the reverse-path information from this command into the reverse-path buffer.

SEND OR MAIL (SOML)

This command is used to initiate a mail transaction in which the mail data is delivered to one or more terminals or mailboxes. For each recipient the mail data is delivered to the recipient's terminal if the recipient is active on the host (and accepting terminal messages), otherwise to the recipient's mailbox. The argument field contains a reverse-path. This command is successful if the message is delivered to a terminal or the mailbox.

The reverse-path consists of an optional list of hosts and the sender mailbox. When the list of hosts is present, it is a "reverse" source route and indicates that the mail was relayed through each host on the list (the first host in the list was the most recent relay). This list is used as a source route to return non-delivery notices to the sender. As each relay host adds itself to the beginning of the list, it must use its name as known in the IPCE to which it is relaying the mail rather than the IPCE from which the mail came (if they are different).

This command clears the reverse-path buffer, the forward-path buffer, and the mail data buffer; and inserts the reverse-path information from this command into the reverse-path buffer.

RFC-821--SMTP Specifications: Commands

SEND AND MAIL (SAML)

This command is used to initiate a mail transaction in which the mail data is delivered to one or more terminals and mailboxes. For each recipient the mail data is delivered to the recipient's terminal if the recipient is active on the host (and accepting terminal messages), and for all recipients to the recipient's mailbox. The argument field contains a reverse-path. This command is successful if the message is delivered to the mailbox.

The reverse-path consists of an optional list of hosts and the sender mailbox. When the list of hosts is present, it is a "reverse" source route and indicates that the mail was relayed through each host on the list (the first host in the list was the most recent relay). This list is used as a source route to return non-delivery notices to the sender. As each relay host adds itself to the beginning of the list, it must use its name as known in the IPCE to which it is relaying the mail rather than the IPCE from which the mail came (if they are different).

This command clears the reverse-path buffer, the forward-path buffer, and the mail data buffer; and inserts the reverse-path information from this command into the reverse-path buffer.

RESET (RSET)

This command specifies that the current mail transaction is to be aborted. Any stored sender, recipients, and mail data must be discarded, and all buffers and state tables cleared. The receiver must send an OK reply.

RFC-821--SMTP Specifications: Commands

VERIFY (VRFY)

This command asks the receiver to confirm that the argument identifies a user. If it is a user name, the full name of the user (if known) and the fully specified mailbox are returned.

This command has no effect on any of the reverse-path buffer, the forward-path buffer, or the mail data buffer.

EXPAND (EXPN)

This command asks the receiver to confirm that the argument identifies a mailing list, and if so, to return the membership of that list. The full name of the users (if known) and the fully specified mailboxes are returned in a multiline reply.

This command has no effect on any of the reverse-path buffer, the forward-path buffer, or the mail data buffer.

HELP (HELP)

This command causes the receiver to send helpful information to the sender of the HELP command. The command may take an argument (e.g., any command name) and return more specific information as a response.

This command has no effect on any of the reverse-path buffer, the forward-path buffer, or the mail data buffer.

NOOP (NOOP)

This command does not affect any parameters or previously entered commands. It specifies no action other than that the receiver send an OK reply.

This command has no effect on any of the reverse-path buffer, the forward-path buffer, or the mail data buffer.

QUIT (QUIT)

This command specifies that the receiver must send an OK reply, and then close the transmission channel.

The receiver should not close the transmission channel until it receives and replies to a QUIT command (even if there was an error). The sender should not close the transmission channel until it send a QUIT command and receives the reply (even if there was an error response to a previous command). If the connection is closed prematurely the receiver should act as if a RSET command had been received (canceling any pending transaction, but not undoing any previously completed transaction), the sender should act as if the command or transaction in progress had received a temporary error (4xx).

TURN (TURN)

This command specifies that the receiver must either (1) send an OK reply and then take on the role of the sender-SMTP, or (2) send a refusal reply and retain the role of the receiver-SMTP.

If program-A is currently the sender-SMTP and it sends the TURN command and receives an OK reply (250) then program-A becomes the receiver-SMTP. Program-A is then in the initial state as if the transmission channel just opened, and it then sends the 220 service ready greeting.

If program-B is currently the receiver-SMTP and it receives the TURN command and sends an OK reply (250) then program-B becomes the sender-SMTP. Program-B is then in the initial state as if the transmission channel just opened, and it then expects to receive the 220 service ready greeting.

To refuse to change roles the receiver sends the 502 reply.

RFC-821--SMTP Specifications: Commands

Command Syntax

The commands consist of a command code followed by an argument field. Command codes are four alphabetic characters. Upper and lower case alphabetic characters are to be treated identically. Thus, any of the following may represent the mail command:

```
MAIL    Mail    mail    MaIl    mAIl
```

This also applies to any symbols representing parameter values, such as "TO" or "to" for the forward-path. Command codes and the argument fields are separated by one or more spaces. However, within the reverse-path and forward-path arguments case is important. In particular, in some hosts the user "smith" is different from the user "Smith".

The argument field consists of a variable length character string ending with the character sequence <CRLF>. The receiver is to take no action until this sequence is received.

Square brackets denote an optional argument field. If the option is not taken, the appropriate default is implied.

The following are the SMTP commands:

```
HELO <SP> <domain> <CRLF>

MAIL <SP> FROM:<reverse-path> <CRLF>

RCPT <SP> TO:<forward-path> <CRLF>

DATA <CRLF>

RSET <CRLF>

SEND <SP> FROM:<reverse-path> <CRLF>

SOML <SP> FROM:<reverse-path> <CRLF>

SAML <SP> FROM:<reverse-path> <CRLF>

VRFY <SP> <string> <CRLF>

EXPN <SP> <string> <CRLF>

HELP [<SP> <string>] <CRLF>

NOOP <CRLF>

QUIT <CRLF>

TURN <CRLF>
```

The syntax of the above argument fields (using BNF notation where applicable) is given below. The "..." notation indicates that a field may be repeated one or more times.

```
<reverse-path> ::= <path>
```

Note: The reverse-path may be empty (null), when server-SMTPs need to send notification messages about problems with notification messages. A null reverse-path is one way to prevent loops in error reporting.

```
<forward-path> ::= <path>

<path> ::= "<" [ <a-d-l> ":" ] <mailbox> ">"

<a-d-l> ::= <at-domain> | <at-domain> "," <a-d-l>

<at-domain> ::= "@" <domain>

<domain> ::=  <element> | <element> "." <domain>

<element> ::= <name> | "#" <number> | "[" <dotnum> "]"

<mailbox> ::= <local-part> "@" <domain>

<local-part> ::= <dot-string> | <quoted-string>

<name> ::= <a> <ldh-str> <let-dig>

<ldh-str> ::= <let-dig-hyp> | <let-dig-hyp> <ldh-str>

<let-dig> ::= <a> | <d>

<let-dig-hyp> ::= <a> | <d> | "-"

<dot-string> ::= <string> | <string> "." <dot-string>

<string> ::= <char> | <char> <string>

<quoted-string> ::=  """ <qtext> """

<qtext> ::=  "\" <x> | "\" <x> <qtext> | <q> | <q> <qtext>

<char> ::= <c> | "\" <x>

<dotnum> ::= <snum> "." <snum> "." <snum> "." <snum>

<number> ::= <d> | <d> <number>

<CRLF> ::= <CR> <LF>

<CR> ::= the carriage return character (ASCII code 13)

<LF> ::= the line feed character (ASCII code 10)

<SP> ::= the space character (ASCII code 32)

<snum> ::= one, two, or three digits representing a decimal
          integer value in the range 0 through 255

<a> ::= any one of the 52 alphabetic characters A through Z
```

```
          in upper case and a through z in lower case

<c> ::= any one of the 128 ASCII characters, but not any
          <special> or <SP>

<d> ::= any one of the ten digits 0 through 9

<q> ::= any one of the 128 ASCII characters except <CR>,
          <LF>, quote ("), or backslash (\)

<x> ::= any one of the 128 ASCII characters (no exceptions)

<special> ::= "<" | ">" | "(" | ")" | "[" | "]" | "\" | "."
          | "," | ";" | ":" | "@"  """ | the control
          characters (ASCII codes 0 through 31 inclusive and
          127)
```

Note that the backslash, "\", is a quote character, which is used to indicate that the next character is to be used literally (instead of its normal interpretation). For example, "Joe\,Smith" could be used to indicate a single nine character user field with comma being the fourth character of the field.

Hosts are generally known by names which are translated to addresses in each host. Note that the name elements of domains are the official names -- no use of nicknames or aliases is allowed.

Sometimes a host is not known to the translation function and communication is blocked. To bypass this barrier two numeric forms are also allowed for host "names". One form is a decimal integer prefixed by a pound sign, "#", which indicates the number is the address of the host. Another form is four small decimal integers separated by dots and enclosed by brackets, e.g., "`[123.255.37.2]`", which indicates a 32-bit ARPA Internet Address in four 8-bit fields.

The time stamp line and the return path line are formally defined as follows:

```
<return-path-line> ::= "Return-Path:" <SP><reverse-path><CRLF>

<time-stamp-line> ::= "Received:" <SP> <stamp> <CRLF>

   <stamp> ::= <from-domain> <by-domain> <opt-info> ";"
             <daytime>

   <from-domain> ::= "FROM" <SP> <domain> <SP>

   <by-domain> ::= "BY" <SP> <domain> <SP>

   <opt-info> ::= [<via>] [<with>] [<id>] [<for>]

   <via> ::= "VIA" <SP> <link> <SP>

   <with> ::= "WITH" <SP> <protocol> <SP>

   <id> ::= "ID" <SP> <string> <SP>

   <for> ::= "FOR" <SP> <path> <SP>
```

```
<link> ::= The standard names for links are registered with
          the Network Information Center.

<protocol> ::= The standard names for protocols are
          registered with the Network Information Center.

<daytime> ::= <SP> <date> <SP> <time>

<date> ::= <dd> <SP> <mon> <SP> <yy>

<time> ::= <hh> ":" <mm> ":" <ss> <SP> <zone>

<dd> ::= the one or two decimal integer day of the month in
          the range 1 to 31.

<mon> ::= "JAN" | "FEB" | "MAR" | "APR" | "MAY" | "JUN" |
          "JUL" | "AUG" | "SEP" | "OCT" | "NOV" | "DEC"

<yy> ::= the two decimal integer year of the century in the
          range 00 to 99.

<hh> ::= the two decimal integer hour of the day in the
          range 00 to 24.

<mm> ::= the two decimal integer minute of the hour in the
          range 00 to 59.

<ss> ::= the two decimal integer second of the minute in the
          range 00 to 59.

<zone> ::= "UT" for Universal Time (the default) or other
          time zone designator (as in RFC-822).
```

Return Path Example

Return-Path:
<@CHARLIE.ARPA,@BAKER.ARPA:JOE@ABLE.ARPA>

Time Stamp Line Example

```
Received: FROM ABC.ARPA BY XYZ.ARPA ; 22 OCT 81 09:23:59 PDT

Received: from ABC.ARPA by XYZ.ARPA via TELENET with X25
id M12345 for Smith@PDQ.ARPA ; 22 OCT 81 09:23:59 PDT
```

SMTP Replies

Replies to SMTP commands are devised to ensure the synchronization of requests and actions in the process of mail transfer, and to guarantee that the sender-SMTP always knows the state of the receiver-SMTP. Every command must generate exactly one reply.

The details of the command-reply sequence are made explicit in the section on Sequencing and State Diagrams.

A receiver-SMTP **should** send only the reply codes listed in Reply Codes in Numeric Order or other standard RFCs. A receiver-SMTP **should** use the text shown in with these codes whenever appropriate.

A sender-SMTP **must** determine its actions only by the reply code, not by the text (except for 251 and 551 replies); any text, including no text at all, must be acceptable. The space (blank) following the reply code is considered part of the text. Whenever possible, a sender-SMTP **should** test only the first digit of the reply code, as specified in Appendix E.

> **Discussion**
> Interoperability problems have arisen with SMTP systems using reply codes that are not listed explicitly in standard RFCs, but are legal according to the theory of reply codes explained in Appendix E.

An SMTP reply consists of a three digit number (transmitted as three alphanumeric characters) followed by some text. The number is intended for use by automata to determine what state to enter next; the text is meant for the human user. It is intended that the three digits contain enough encoded information that the sender-SMTP need not examine the text and may either discard it or pass it on to the user, as appropriate. In particular, the text may be receiver-dependent and context dependent, so there are likely to be varying texts for each reply code. A discussion of the theory of reply codes is given in Appendix E. Formally, a reply is defined to be the sequence: a three-digit code, <SP>, one line of text, and <CRLF>, or a multiline reply (as defined in Appendix E). Only the EXPN and HELP commands are expected to result in multiline replies in normal circumstances, however multiline replies are allowed for any command.

RFC-821--SMTP Specifications

Reply Codes By Function Groups

500 Syntax error, command unrecognized [This may include errors such as command line too long]
501 Syntax error in parameters or arguments
502 Command not implemented
503 Bad sequence of commands
504 Command parameter not implemented

211 System status, or system help reply
214 Help message
[Information on how to use the receiver or the meaning of a particular non-standard command; this reply is useful only to the human user]

220 <domain> Service ready
221 <domain> Service closing transmission channel
421 <domain> Service not available, closing transmission channel [This may be a reply to any command if the service knows it must shut down]

250 Requested mail action okay, completed
251 User not local; will forward to <forward-path>
252 Cannot VRFY user (e.g., info is not local), but will take message for this user and attempt delivery.
450 Requested mail action not taken: mailbox unavailable [E.g., mailbox busy]
550 Requested action not taken: mailbox unavailable [E.g., Mailbox not found, no access]
451 Requested action aborted: error in processing
551 User not local; please try <forward-path>
452 Requested action not taken: insufficient system storage
552 Requested mail action aborted: exceeded storage allocation
553 Requested action not taken: mailbox name not allowed [E.g., mailbox syntax incorrect]
354 Start mail input; end with <CRLF>.<CRLF>
554 Transaction failed

Numeric Order List Of Reply Codes

211 System status, or system help reply
214 Help message [Information on how to use the receiver or the meaning of a particular non-standard command; this reply is useful only to the human user]
220 <domain> Service ready
221 <domain> Service closing transmission channel
250 Requested mail action okay, completed
251 User not local; will forward to <forward-path>
252 Cannot VRFY user (e.g., info is not local), but will take message for this user and attempt delivery.

354 Start mail input; end with <CRLF>.<CRLF>

421 <domain> Service not available, closing transmission channel. [This may be a reply to any command if the service knows it must shut down]
450 Requested mail action not taken: mailbox unavailable [E.g., mailbox busy]
451 Requested action aborted: local error in processing
452 Requested action not taken: insufficient system storage

500 Syntax error, command unrecognized [This may include errors such as command line too long]
501 Syntax error in parameters or arguments
502 Command not implemented
503 Bad sequence of commands
504 Command parameter not implemented
550 Requested action not taken: mailbox unavailable [E.g., mailbox not found, no access]
551 User not local; please try <forward-path>
552 Requested mail action aborted: exceeded storage allocation
553 Requested action not taken: mailbox name not allowed [E.g., mailbox syntax incorrect]
554 Transaction failed

RFC-821--SMTP Specifications

Sequencing Of Commands and Replies

The communication between the sender and receiver is intended to be an alternating dialogue, controlled by the sender. As such, the sender issues a command and the receiver responds with a reply. The sender must wait for this response before sending further commands.

One important reply is the connection greeting. Normally, a receiver will send a 220 "Service ready" reply when the connection is completed. The sender should wait for this greeting message before sending any commands.

Note: all the greeting type replies have the official name of the server host as the first word following the reply code.

For example,

```
220 <SP> USC-ISIF.ARPA <SP> Service ready <CRLF>
```

The table below lists alternative success and failure replies for each command. These must be strictly adhered to; a receiver may substitute text in the replies, but the meaning and action implied by the code numbers and by the specific command reply sequence cannot be altered.

COMMAND-REPLY SEQUENCES

Each command is listed with its possible replies. The prefixes used before the possible replies are "P" for preliminary (not used in SMTP), "I" for intermediate, "S" for success, "F" for failure, and "E" for error. The 421 reply (service not available, closing transmission channel) may be given to any command if the SMTP-receiver knows it must shut down. This listing forms the basis for the State Diagrams.

```
CONNECTION ESTABLISHMENT
S: 220
F: 421 HELO
               S: 250
               E: 500, 501, 504, 421
MAIL
               S: 250
               F: 552, 451, 452
               E: 500, 501, 421

RCPT
               S: 250, 251
               F: 550, 551, 552, 553, 450, 451, 452
               E: 500, 501, 503, 421
DATA
               I: 354 -> data -> S: 250
                                 F: 552, 554, 451, 452
               F: 451, 554
               E: 500, 501, 503, 421
RSET
               S: 250
               E: 500, 501, 504, 421
```

SEND

S: 250
F: 552, 451, 452
E: 500, 501, 502, 421

SOML

S: 250
F: 552, 451, 452
E: 500, 501, 502, 421

SAML

S: 250
F: 552, 451, 452
E: 500, 501, 502, 421

VRFY

S: 250, 251
F: 550, 551, 553
E: 500, 501, 502, 504, 421

EXPN

S: 250
F: 550
E: 500, 501, 502, 504, 421

HELP

S: 211, 214
E: 500, 501, 502, 504, 421

NOOP

S: 250
E: 500, 421

QUIT

S: 221
E: 500

TURN

S: 250
F: 502
E: 500, 503

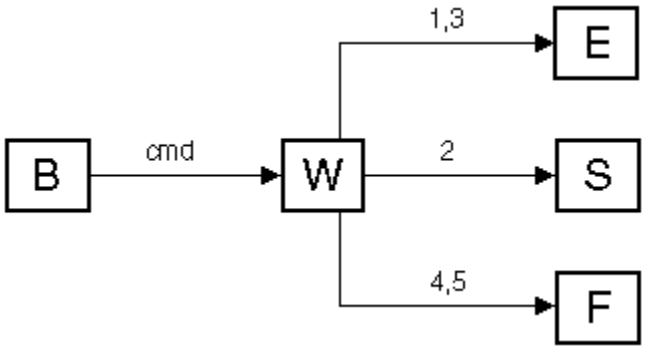
RFC-821--SMTP Specifications

State Diagrams

Following are state diagrams for a simple-minded SMTP implementation. Only the first digit of the reply codes is used. There is one state diagram for each group of SMTP commands. The command groupings were determined by constructing a model for each command and then collecting together the commands with structurally identical models.

For each command there are three possible outcomes: "success" (S), "failure" (F), and "error" (E). In the state diagrams below we use the symbol B for "begin", and the symbol W for "wait for reply".

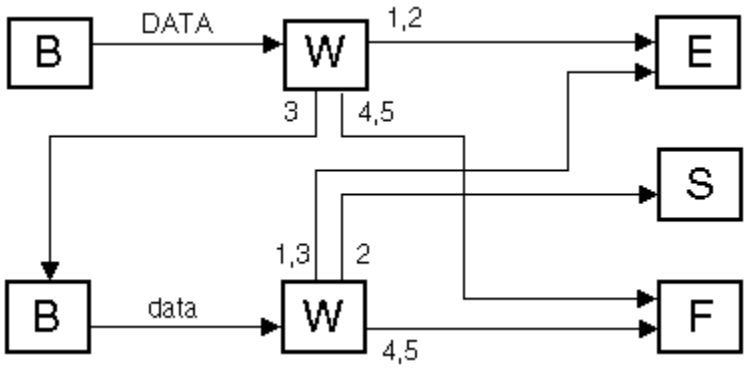
First, the diagram that represents most of the SMTP commands:



This diagram models the commands:

HELO, MAIL, RCPT, RSET, SEND, SOML, SAML, VRFY, EXPN, HELP, NOOP, QUIT, TURN.

A more complex diagram models the DATA command:



Note that the "data" here is a series of lines sent from the sender to the receiver with no response expected until the last line is sent.

Minimum Implementation

In order to make SMTP workable, the following minimum implementation is required for all receivers:

COMMANDS -- HELO
MAIL
RCPT
DATA
RSET
NOOP
QUIT
VRFY

Transparency

Without some provision for data transparency the character sequence "<CRLF>.<CRLF>" ends the mail text and cannot be sent by the user. In general, users are not aware of such "forbidden" sequences. To allow all user composed text to be transmitted transparently the following procedures are used.

1. Before sending a line of mail text the sender-SMTP checks the first character of the line. If it is a period, one additional period is inserted at the beginning of the line.

2. When a line of mail text is received by the receiver-SMTP it checks the line. If the line is composed of a single period it is the end of mail. If the first character is a period and there are other characters on the line, the first character is deleted.

Implementors **must** be sure that their mail systems always add and delete periods to ensure message transparency.

The mail data may contain any of the 128 ASCII characters. All characters are to be delivered to the recipient's mailbox including format effectors and other control characters. If the transmission channel provides an 8-bit byte (octets) data stream, the 7-bit ASCII codes are transmitted right justified in the octets with the high order bits cleared to zero.

In some systems it may be necessary to transform the data as it is received and stored. This may be necessary for hosts that use a different character set than ASCII as their local character set, or that store data in records rather than strings. If such transforms are necessary, they must be reversible -- especially if such transforms are applied to mail being relayed.

Sizes

There are several objects that have required minimum maximum sizes. That is, every implementation must be able to receive objects of at least these sizes, but must not send objects larger than these sizes.

```
**************************************************
*                                                *
*  TO THE MAXIMUM EXTENT POSSIBLE, IMPLEMENTATION  *
*  TECHNIQUES WHICH IMPOSE NO LIMITS ON THE LENGTH *
*  OF THESE OBJECTS SHOULD BE USED.                *
*                                                *
**************************************************
```

user

The maximum total length of a user name is 64 characters.

domain

The maximum total length of a domain name or number is 64 characters.

path

The maximum total length of a reverse-path or forward-path is 256 characters (including the punctuation and element separators).

command line

The maximum total length of a command line including the command word and the <CRLF> is 512 characters.

reply line

The maximum total length of a reply line including the reply code and the <CRLF> is 512 characters.

text line

The maximum total length of a text line including the <CRLF> is 1000 characters (but not counting the leading dot duplicated for transparency).

recipients buffer

The maximum total number of recipients that must be buffered is 100 recipients.

Errors due to exceeding these limits may be reported by using the reply codes, for example:

500 Line too long.

501 Path too long

552 Too many recipients.

552 Too much mail data.

Appendix A: TCP Transport Service

The Transmission Control Protocol is used in the ARPA Internet, and in any network following the US DoD standards for internetwork protocols.

Connection Establishment

The SMTP transmission channel is a TCP connection established between the sender process port U and the receiver process port L. This single full duplex connection is used as the transmission channel. This protocol is assigned the service port 25 (31 octal), that is L=25.

Data Transfer

The TCP connection supports the transmission of 8-bit bytes. The SMTP data is 7-bit ASCII characters. Each character is transmitted as an 8-bit byte with the high-order bit cleared to zero.

Appendix B: NCP Transport Service

The ARPANET Host-to-Host Protocol (implemented by the Network Control Program) may be used in the ARPANET.

Connection Establishment

The SMTP transmission channel is established via NCP between the sender process socket U and receiver process socket L. The Initial Connection Protocol is followed resulting in a pair of simplex connections. This pair of connections is used as the transmission channel. This protocol is assigned the contact socket 25 (31 octal), that is L=25.

Data Transfer

The NCP data connections are established in 8-bit byte mode. The SMTP data is 7-bit ASCII characters. Each character is transmitted as an 8-bit byte with the high-order bit cleared to zero.

Appendix C: NITS

The Network Independent Transport Service may be used.

Connection Establishment

The SMTP transmission channel is established via NITS between the sender process and receiver process. The sender process executes the CONNECT primitive, and the waiting receiver process executes the ACCEPT primitive.

Data Transfer

The NITS connection supports the transmission of 8-bit bytes. The SMTP data is 7-bit ASCII characters. Each character is transmitted as an 8-bit byte with the high-order bit cleared to zero.

Appendix D: X.25 Transport Service

It may be possible to use the X.25 service as provided by the Public Data Networks directly, however, it is suggested that a reliable end-to-end protocol such as TCP be used on top of X.25 connections.

Appendix E: Theory Of Reply Codes

The three digits of the reply each have a special significance. The first digit denotes whether the response is good, bad or incomplete. An unsophisticated sender-SMTP will be able to determine its next action (proceed as planned, redo, retrench, etc.) by simply examining this first digit. A sender-SMTP that wants to know approximately what kind of error occurred (e.g., mail system error, command syntax error) may examine the second digit, reserving the third digit for the finest gradation of information.

There are five values for the first digit of the reply code:

1yz Positive Preliminary reply

The command has been accepted, but the requested action is being held in abeyance, pending confirmation of the information in this reply. The sender-SMTP should send another command specifying whether to continue or abort the action.

[Note: SMTP does not have any commands that allow this type of reply, and so does not have the continue or abort commands.]

2yz Positive Completion reply

The requested action has been successfully completed. A new request may be initiated.

3yz Positive Intermediate reply

The command has been accepted, but the requested action is being held in abeyance, pending receipt of further information. The sender-SMTP should send another command specifying this information. This reply is used in command sequence groups.

4yz Transient Negative Completion reply

The command was not accepted and the requested action did not occur. However, the error condition is temporary and the action may be requested again. The sender should return to the beginning of the command sequence (if any). It is difficult to assign a meaning to "transient" when two different sites (receiver- and sender-SMTPs) must agree on the interpretation. Each reply in this category might have a different time value, but the sender-SMTP is encouraged to try again. A rule of thumb to determine if a reply fits into the 4yz or the 5yz category (see below) is that replies are 4yz if they can be repeated without any change in command form or in properties of the sender or receiver. (E.g., the command is repeated identically and the receiver does not put up a new implementation.)

5yz Permanent Negative Completion reply

The command was not accepted and the requested action did not occur. The sender-SMTP is discouraged from repeating the exact request (in the same sequence). Even some "permanent" error conditions can be corrected, so the human user may want to direct the sender-SMTP to reinitiate the command sequence by direct action at some point in the future (e.g., after the spelling has been changed, or the user has altered the account status).

The second digit encodes responses in specific categories:

x0z **Syntax** -- These replies refer to syntax errors, syntactically correct commands that don't fit any functional category, and unimplemented or superfluous commands.

x1z **Information** -- These are replies to requests for information, such as status or help.

x2z **Connections** -- These are replies referring to the transmission channel.

x3z Unspecified as yet.

x4z Unspecified as yet.

x5z **Mail system** -- These replies indicate the status of the receiver mail system vis-a-vis the requested transfer or other mail system action.

The third digit gives a finer gradation of meaning in each category specified by the second digit. The list of replies illustrates this. Each reply text is recommended rather than mandatory, and may even change according to the command with which it is associated. On the other hand, the reply codes must strictly follow the specifications in this section. Receiver implementations should not invent new codes for slightly different situations from the ones described here, but rather adapt codes already defined.

For example, a command such as NOOP whose successful execution does not offer the sender-SMTP any new information will return a 250 reply. The response is 502 when the command requests an unimplemented non-site-specific action. A refinement of that is the 504 reply for a command that is implemented, but that requests an unimplemented parameter.

The reply text may be longer than a single line; in these cases the complete text must be marked so the sender-SMTP knows when it can stop reading the reply. This requires a special format to indicate a multiple line reply.

The format for multiline replies requires that every line, except the last, begin with the reply code, followed immediately by a hyphen, "-" (also known as minus), followed by text. The last line will begin with the reply code, followed immediately by <SP>, optionally some text, and <CRLF>.

For example:

```
123-First line
123-Second line
123-234 text beginning with numbers
123 The last line
```

In many cases the sender-SMTP then simply needs to search for the reply code followed by <SP> at the beginning of a line, and ignore all preceding lines. In a few cases, there is important data for the sender in the reply "text". The sender will know these cases from the current context.

Appendix F: Scenarios

This section presents complete scenarios of several types of SMTP sessions.

A Typical SMTP Transaction Scenario

This SMTP example shows mail sent by Smith at host USC-ISIF, to Jones, Green, and Brown at host BBN-UNIX. Here we assume that host USC-ISIF contacts host BBN-UNIX directly. The mail is accepted for Jones and Brown. Green does not have a mailbox at host BBN-UNIX.

```
R: 220 BBN-UNIX.ARPA Simple Mail Transfer Service Ready
S: HELO USC-ISIF.ARPA
R: 250 BBN-UNIX.ARPA

S: MAIL FROM:<Smith@USC-ISIF.ARPA>
R: 250 OK

S: RCPT TO:<Jones@BBN-UNIX.ARPA>
R: 250 OK

S: RCPT TO:<Green@BBN-UNIX.ARPA>
R: 550 No such user here

S: RCPT TO:<Brown@BBN-UNIX.ARPA>
R: 250 OK

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: Blah blah blah...
S: ...etc. etc. etc.
S: .
R: 250 OK

S: QUIT
R: 221 BBN-UNIX.ARPA Service closing transmission channel
```

Aborted SMTP Transaction Scenario

```
R: 220 MIT-Multics.ARPA Simple Mail Transfer Service Ready
S: HELO ISI-VAXA.ARPA
R: 250 MIT-Multics.ARPA

S: MAIL FROM:<Smith@ISI-VAXA.ARPA>
R: 250 OK

S: RCPT TO:<Jones@MIT-Multics.ARPA>
R: 250 OK

S: RCPT TO:<Green@MIT-Multics.ARPA>
R: 550 No such user here

S: RSET
R: 250 OK

S: QUIT
R: 221 MIT-Multics.ARPA Service closing transmission channel
```

Relayed Mail Scenario

Step 1 -- Source Host to Relay Host

```
R: 220 USC-ISIE.ARPA Simple Mail Transfer Service Ready
S: HELO MIT-AI.ARPA
R: 250 USC-ISIE.ARPA

S: MAIL FROM:<JQP@MIT-AI.ARPA>
R: 250 OK

S: RCPT TO:<@USC-ISIE.ARPA:Jones@BBN-VAX.ARPA>
R: 250 OK

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: Date: 2 Nov 81 22:33:44
S: From: John Q. Public <JQP@MIT-AI.ARPA>
S: Subject:  The Next Meeting of the Board
S: To: Jones@BBN-Vax.ARPA
S:
S: Bill:
S: The next meeting of the board of directors will be
S: on Tuesday.
S:                                                John.
S: .
R: 250 OK

S: QUIT
R: 221 USC-ISIE.ARPA Service closing transmission channel
```

Step 2 -- Relay Host to Destination Host

```
R: 220 BBN-VAX.ARPA Simple Mail Transfer Service Ready
S: HELO USC-ISIE.ARPA
R: 250 BBN-VAX.ARPA

S: MAIL FROM:<@USC-ISIE.ARPA:JQP@MIT-AI.ARPA>
R: 250 OK

S: RCPT TO:<Jones@BBN-VAX.ARPA>
R: 250 OK

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: Received: from MIT-AI.ARPA by USC-ISIE.ARPA ;
   2 Nov 81 22:40:10 UT
S: Date: 2 Nov 81 22:33:44
S: From: John Q. Public <JQP@MIT-AI.ARPA>
S: Subject:  The Next Meeting of the Board
S: To: Jones@BBN-Vax.ARPA
```

```
S:
S: Bill:
S: The next meeting of the board of directors will be
S: on Tuesday.
S:                                                  John.
S: .
R: 250 OK

S: QUIT
R: 221 USC-ISIE.ARPA Service closing transmission channel
```

Verifying and Sending Scenario

```
R: 220 SU-SCORE.ARPA Simple Mail Transfer Service Ready
S: HELO MIT-MC.ARPA
R: 250 SU-SCORE.ARPA

S: VRFY Crispin
R: 250 Mark Crispin <Admin.MRC@SU-SCORE.ARPA>

S: SEND FROM:<EAK@MIT-MC.ARPA>
R: 250 OK

S: RCPT TO:<Admin.MRC@SU-SCORE.ARPA>
R: 250 OK

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: Blah blah blah...
S: ...etc. etc. etc.
S: .
R: 250 OK

S: QUIT
R: 221 SU-SCORE.ARPA Service closing transmission channel
```

Sending and Mailing Scenario

First the user's name is verified, then an attempt is made to send to the user's terminal. When that fails, the messages is mailed to the user's mailbox.

```
R: 220 SU-SCORE.ARPA Simple Mail Transfer Service Ready
S: HELO MIT-MC.ARPA
R: 250 SU-SCORE.ARPA

S: VRFY Crispin
R: 250 Mark Crispin <Admin.MRC@SU-SCORE.ARPA>

S: SEND FROM:<EAK@MIT-MC.ARPA>
R: 250 OK

S: RCPT TO:<Admin.MRC@SU-SCORE.ARPA>
R: 450 User not active now

S: RSET
R: 250 OK

S: MAIL FROM:<EAK@MIT-MC.ARPA>
R: 250 OK

S: RCPT TO:<Admin.MRC@SU-SCORE.ARPA>
R: 250 OK

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: Blah blah blah...
S: ...etc. etc. etc.
S: .
R: 250 OK

S: QUIT
R: 221 SU-SCORE.ARPA Service closing transmission channel
```

More Efficient Sending and Mailing Scenario

A more efficient way to do the preceding scenario.

```
R: 220 SU-SCORE.ARPA Simple Mail Transfer Service Ready
S: HELO MIT-MC.ARPA
R: 250 SU-SCORE.ARPA

S: VRFY Crispin
R: 250 Mark Crispin <Admin.MRC@SU-SCORE.ARPA>

S: SOML FROM:<EAK@MIT-MC.ARPA>
R: 250 OK

S: RCPT TO:<Admin.MRC@SU-SCORE.ARPA>
R: 250 User not active now, so will do mail.

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: Blah blah blah...
S: ...etc. etc. etc.
S: .
R: 250 OK

S: QUIT
R: 221 SU-SCORE.ARPA Service closing transmission channel
```

Mailing List Scenario

First each of two mailing lists are expanded in separate sessions with different hosts. Then the message is sent to everyone that appeared on either list (but no duplicates) via a relay host.

Step 1 -- Expanding the First List

```
R: 220 MIT-AI.ARPA Simple Mail Transfer Service Ready
S: HELO SU-SCORE.ARPA
R: 250 MIT-AI.ARPA

S: EXPN Example-People
R: 250-<ABC@MIT-MC.ARPA>
R: 250-Fred Fonebone <Fonebone@USC-ISIQ.ARPA>
R: 250-Xenon Y. Zither <XYZ@MIT-AI.ARPA>
R: 250-Quincy Smith <@USC-ISIF.ARPA:Q-Smith@ISI-VAXA.ARPA>
R: 250-<joe@foo-unix.ARPA>
R: 250 <xyz@bar-unix.ARPA>

S: QUIT
R: 221 MIT-AI.ARPA Service closing transmission channel
```

Step 2 -- Expanding the Second List

```
R: 220 MIT-MC.ARPA Simple Mail Transfer Service Ready
S: HELO SU-SCORE.ARPA
R: 250 MIT-MC.ARPA

S: EXPN Interested-Parties
R: 250-Al Calico <ABC@MIT-MC.ARPA>
R: 250-<XYZ@MIT-AI.ARPA>
R: 250-Quincy Smith <@USC-ISIF.ARPA:Q-Smith@ISI-VAXA.ARPA>
R: 250-<fred@BBN-UNIX.ARPA>
R: 250 <xyz@bar-unix.ARPA>

S: QUIT
R: 221 MIT-MC.ARPA Service closing transmission channel
```

Step 3 -- Mailing to All via a Relay Host

```
R: 220 USC-ISIE.ARPA Simple Mail Transfer Service Ready
S: HELO SU-SCORE.ARPA
R: 250 USC-ISIE.ARPA

S: MAIL FROM:<Account.Person@SU-SCORE.ARPA>
R: 250 OK
S: RCPT TO:<@USC-ISIE.ARPA:ABC@MIT-MC.ARPA>
R: 250 OK
S: RCPT TO:<@USC-ISIE.ARPA:Fonebone@USC-ISIQA.ARPA>
```

```
R: 250 OK
S: RCPT TO:<@USC-ISIE.ARPA:XYZ@MIT-AI.ARPA>
R: 250 OK
S: RCPT
    TO:<@USC-ISIE.ARPA,@USC-ISIF.ARPA:Q-Smith@ISI- VAXA.ARPA>
R: 250 OK
S: RCPT TO:<@USC-ISIE.ARPA:joe@FOO-UNIX.ARPA>
R: 250 OK
S: RCPT TO:<@USC-ISIE.ARPA:xyz@BAR-UNIX.ARPA>
R: 250 OK
S: RCPT TO:<@USC-ISIE.ARPA:fred@BBN-UNIX.ARPA>
R: 250 OK

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: Blah blah blah...
S: ...etc. etc. etc.
S: .
R: 250 OK

S: QUIT
R: 221 USC-ISIE.ARPA Service closing transmission channel
```

Simple Forwarding Scenario

```
R: 220 USC-ISIF.ARPA Simple Mail Transfer Service Ready
S: HELO LBL-UNIX.ARPA
R: 250 USC-ISIF.ARPA

S: MAIL FROM:<mo@LBL-UNIX.ARPA>
R: 250 OK

S: RCPT TO:<fred@USC-ISIF.ARPA>
R: 251 User not local; will forward to <Jones@USC-ISI.ARPA>

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: Blah blah blah...
S: ...etc. etc. etc.
S: .
R: 250 OK

S: QUIT
R: 221 USC-ISIF.ARPA Service closing transmission channel
```

Complex Forwarding Scenario

Step 1 -- Trying the Mailbox at the First Host

```
R: 220 USC-ISIF.ARPA Simple Mail Transfer Service Ready
S: HELO LBL-UNIX.ARPA
R: 250 USC-ISIF.ARPA

S: MAIL FROM:<mo@LBL-UNIX.ARPA>
R: 250 OK

S: RCPT TO:<fred@USC-ISIF.ARPA>
R: 251 User not local; will forward to <Jones@USC-ISI.ARPA>

S: RSET
R: 250 OK

S: QUIT
R: 221 USC-ISIF.ARPA Service closing transmission channel
```

Step 2 -- Delivering the Mail at the Second Host

```
R: 220 USC-ISI.ARPA Simple Mail Transfer Service Ready
S: HELO LBL-UNIX.ARPA
R: 250 USC-ISI.ARPA

S: MAIL FROM:<mo@LBL-UNIX.ARPA>
R: 250 OK

S: RCPT TO:<Jones@USC-ISI.ARPA>
R: OK

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: Blah blah blah...
S: ...etc. etc. etc.
S: .
R: 250 OK

S: QUIT
R: 221 USC-ISI.ARPA Service closing transmission channel
```

Too Many Recipients Scenario

```
R: 220 BERKELEY.ARPA Simple Mail Transfer Service Ready
S: HELO USC-ISIF.ARPA
R: 250 BERKELEY.ARPA

S: MAIL FROM:<Postel@USC-ISIF.ARPA>
R: 250 OK

S: RCPT TO:<fabry@BERKELEY.ARPA>
R: 250 OK

S: RCPT TO:<eric@BERKELEY.ARPA>
R: 552 Recipient storage full, try again in another transaction

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: Blah blah blah...
S: ...etc. etc. etc.
S: .
R: 250 OK

S: MAIL FROM:<Postel@USC-ISIF.ARPA>
R: 250 OK

S: RCPT TO:<eric@BERKELEY.ARPA>
R: 250 OK

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: Blah blah blah...
S: ...etc. etc. etc.
S: .
R: 250 OK

S: QUIT
R: 221 BERKELEY.ARPA Service closing transmission channel
```

Note that a real implementation must handle as many recipients as specified in the Specification of Sizes section.

GLOSSARY

ASCII
Command
Domain
End Of Mail Data Indication
Host
Line
Mail Data
Mailbox
Receiver-Smtp Process
Reply
Sender-Smtp Process
Session
Transaction
Transmission
Transport
User
Word
<CRLF>
<SP>

American Standard Code for Information Interchange.

A request for a mail service action sent by the sender-SMTP to the receiver-SMTP.

The hierarchially structured global character string address of a host computer in the mail system.

A special sequence of characters that indicates the end of the mail data. In particular, the five characters carriage return, line feed, period, carriage return, line feed, in that order.

A computer in the internetwork environment on which mailboxes or SMTP processes reside.

A a sequence of ASCII characters ending with a <CRLF>.

A sequence of ASCII characters of arbitrary length, which conforms to the standard set in the Standard for the Format of ARPA Internet Text Messages (RFC-822).

A character string (address) which identifies a user to whom mail is to be sent. Mailbox normally consists of the host and user specifications. The standard mailbox naming convention is defined to be "user@domain". Additionally, the "container" in which mail is stored.

A process which transfers mail in cooperation with a sender-SMTP process. It waits for a connection to be established via the transport service. It receives SMTP commands from the sender-SMTP, sends replies, and performs the specified operations.

A reply is an acknowledgment (positive or negative) sent from receiver to sender via the transmission channel in response to a command. The general form of a reply is a completion code (including error codes) followed by a text string. The codes are for use by programs and the text is usually intended for human users.

A process which transfers mail in cooperation with a receiver-SMTP process. A local language may be used in the user interface command/reply dialogue. The sender-SMTP initiates the transport service connection. It initiates SMTP commands, receives replies, and governs the transfer of mail.

The set of exchanges that occur while the transmission channel is open.

The set of exchanges required for one message to be transmitted for one or more recipients.

A full-duplex communication path between a sender-SMTP and a receiver-SMTP for the exchange of commands, replies, and mail text.

Any reliable stream-oriented data communication services. For example, NCP, TCP, NITS.

A human being (or a process on behalf of a human being) wishing to obtain mail transfer service. In addition, a recipient of computer mail.

A sequence of printing characters.

The characters carriage return and line feed (in that order).

The space character.

ASCII, "USA Code for Information Interchange", United States of America Standards Institute, X3.4, 1968. Also in: Feinler, E. and J. Postel, eds., "ARPANET Protocol Handbook", NIC 7104, for the Defense Communications Agency by SRI International, Menlo Park, California, Revised January 1978.

Postel, J., ed., "Transmission Control Protocol - DARPA Internet Program Protocol Specification", RFC 793, USC/Information Sciences Institute, NTIS AD Number A111091, September 1981. Also in: Feinler, E. and J. Postel, eds., "Internet Protocol Transition Workbook", SRI International, Menlo Park, California, March 1982.

McKenzie,A., "Host/Host Protocol for the ARPA Network", NIC 8246, January 1972. Also in: Feinler, E. and J. Postel, eds., "ARPANET Protocol Handbook", NIC 7104, for the Defense Communications Agency by SRI International, Menlo Park, California, Revised January 1978.

Postel, J., "Official Initial Connection Protocol", NIC 7101, 11 June 1971. Also in: Feinler, E. and J. Postel, eds., "ARPANET Protocol Handbook", NIC 7104, for the Defense Communications Agency by SRI International, Menlo Park, California, Revised January 1978.

PSS/SG3, "A Network Independent Transport Service", Study Group 3, The Post Office PSS Users Group, February 1980. Available from the DCPU, National Physical Laboratory, Teddington, UK.

CCITT, "Recommendation X.25 - Interface Between Data Terminal Equipment (DTE) and Data Circuit-terminating Equipment (DCE) for Terminals Operating in the Packet Mode on Public Data Networks," CCITT Orange Book, Vol. VIII.2, International Telephone and Telegraph Consultative Committee, Geneva, 1976.

–

RFC-822 STANDARD FOR THE FORMAT OF ARPA INTERNET TEXT MESSAGES (Obsoletes: RFC-733)

David H. Crocker
August 13, 1982
Dept. of Electrical Engineering
University of Delaware, Newark, DE 19711
Network: DCrocker@UDel-Relay

Preface

By 1977, the Arpanet employed several informal standards for the text messages (mail) sent among its host computers. It was felt necessary to codify these practices and provide for those features that seemed imminent. The result of that effort was Request for Comments RFC-733, "Standard for the Format of ARPA Network Text Message", by Crocker, Vittal, Pogran, and Henderson. The specification attempted to avoid major changes in existing software, while permitting several new features.

This document revises the specifications in RFC-733, in order to serve the needs of the larger and more complex ARPA Internet. Some of RFC-733's features failed to gain adequate acceptance. In order to simplify the standard and the software that follows it, these features have been removed. A different addressing scheme is used, to handle the case of inter-network mail; and the concept of re-transmission has been introduced.

This specification is intended for use in the ARPA Internet. However, an attempt has been made to free it of any dependence on that environment, so that it can be applied to other network text message systems.

The specification of RFC-733 took place over the course of one year, using the ARPANET mail environment, itself, to provide an on-going forum for discussing the capabilities to be included. More than twenty individuals, from across the country, participated in the original discussion. The development of this revised specification has, similarly, utilized network mail-based group discussion. Both specification efforts greatly benefited from the comments and ideas of the participants.

The syntax of the standard, in RFC-733, was originally specified in the Backus-Naur Form (BNF) meta-language Ken L. Harrenstien, of SRI International, was responsible for re-coding the BNF into an augmented BNF that makes the representation smaller and easier to understand.

Scope

This standard specifies a syntax for text messages that are sent among computer users, within the framework of "electronic mail". The standard supersedes the one specified in ARPANET Request for Comments 733, "Standard for the Format of ARPA Network Text Messages".

In this context, messages are viewed as having an envelope and contents. The envelope contains whatever information is needed to accomplish transmission and delivery. The contents compose the object to be delivered to the recipient. This standard applies only to the format and some of the semantics of message contents. It contains no specification of the information in the envelope.

However, some message systems may use information from the contents to create the envelope. It is intended that this standard facilitate the acquisition of such information by programs.

Some message systems may store messages in formats that differ from the one specified in this standard. This specification is intended strictly as a definition of what message content format is to be passed BETWEEN hosts.

Note: This standard is NOT intended to dictate the internal formats used by sites, the specific message system features that they are expected to support, or any of the characteristics of user interface programs that create or read messages.

A distinction should be made between what the specification REQUIRES and what it ALLOWS. Messages can be made complex and rich with formally-structured components of information or can be kept small and simple, with a minimum of such information. Also, the standard simplifies the interpretation of differing visual formats in messages; only the visual aspect of a message is affected and not the interpretation of information within it. Implementors may choose to retain such visual distinctions.

The formal definition is divided into four levels. The bottom level describes the meta-notation used in this document. The second level describes basic lexical analyzers that feed tokens to higher-level parsers. Next is an overall specification for messages; it permits distinguishing individual fields. Finally, there is definition of the contents of several structured fields.

Communication Framework

Messages consist of lines of text. No special provisions are made for encoding drawings, facsimile, speech, or structured text. No significant consideration has been given to questions of data compression or to transmission and storage efficiency, and the standard tends to be free with the number of bits consumed. For example, field names are specified as free text, rather than special terse codes.

A general "memo" framework is used. That is, a message consists of some information in a rigid format, followed by the main part of the message, with a format that is not specified in this document. The syntax of several fields of the rigidly-formated ("headers") section is defined in this specification; some of these fields must be included in all messages.

The syntax that distinguishes between header fields is specified separately from the internal syntax for particular fields. This separation is intended to allow simple parsers to operate on the general structure of messages, without concern for the detailed structure of individual header fields. Appendix B is provided to facilitate construction of these parsers.

In addition to the fields specified in this document, it is expected that other fields will gain common use. As necessary, the specifications for these "extension-fields" will be published through the same mechanism used to publish this document. Users may also wish to extend the set of fields that they use privately. Such "user-defined fields" are permitted.

The framework severely constrains document tone and appearance and is primarily useful for most intra-organization communications and well-structured inter-organization communication. It also can be used for some types of inter-process communication, such as simple file transfer and remote job entry. A more robust framework might allow for multi-font, multi-color, multi-dimension encoding of information. A less robust one, as is present in most single-machine message systems, would more severely constrain the ability to add fields and the decision to include specific fields. In contrast with paper-based communication, it is interesting to note that the RECEIVER of a message can exercise an extraordinary amount of control over the message's appearance. The amount of actual control available to message receivers is contingent upon the capabilities of their individual message systems.

Notational Conventions

This specification uses an augmented Backus-Naur Form (BNF) notation. The differences from standard BNF involve naming rules and indicating repetition and "local" alternatives.

Rule Naming

Angle brackets ("<", ">") are not used, in general. The name of a rule is simply the name itself, rather than "<name>". Quotation-marks enclose literal text (which may be upper and/or lower case). Certain basic rules are in uppercase, such as SPACE, TAB, CRLF, DIGIT, ALPHA, etc. Angle brackets are used in rule definitions, and in the rest of this document, whenever their presence will facilitate discerning the use of rule names.

Rule1 / Rule2: Alternatives

Elements separated by slash ("/") are alternatives. Therefore "foo / bar" will accept foo or bar.

(Rule1 Rule2): Local Alternatives

Elements enclosed in parentheses are treated as a single element. Thus, "(elem (foo / bar) elem)" allows the token sequences "elem foo elem" and "elem bar elem".

*Rule: Repetition

The character "*" preceding an element indicates repetition. The full form is:

```
<l>*<m>element
```

indicating at least <l> and at most <m> occurrences of element. Default values are 0 and infinity so that "*(element)" allows any number, Including zero; "1*element" requires at least one; and "1*2element" allows one or two.

[Rule]: Optional

Square brackets enclose optional elements; "[foo bar]" is equivalent to "*1(foo bar)".

Nrule: Specific Repetition

"<n>(element)" is equivalent to "<n>*<n>(element)"; that is, exactly <n> occurrences of (element). Thus 2DIGIT is a 2-digit number, and 3ALPHA is a string of three alphabetic characters.

#Rule: Lists

A construct "#" is defined, similar to "*", as follows:

```
<l>#<m>element
```

indicating at least <l> and at most <m> elements, each separated by one or more commas (","). This makes the usual form of lists very easy; a rule such as '(element *("," element))' can be shown as "1#element". Wherever this construct is used, null elements are allowed, but do not contribute to the count of elements present. That is, "(element),,(element)" is permitted, but counts as only two elements. Therefore, where at least one element is required, at least one non-null element must be present. Default values are 0 and infinity so that "#(element)" allows any number, including zero; "1#element" requires at least one; and "1#2element" allows one or two.

Comments

A semi-colon, set off some distance to the right of rule text, starts a comment that continues to the end of line. This is a simple way of including useful notes in parallel with the specifications.

RFC-822 Standard for ARPA Internet Text Messages:

Lexical Analysis Of Messages

RFC-822 Standard for ARPA Internet Text Messages:

Lexical Analysis Of Messages: General Description

A message consists of header fields and, optionally, a body. The body is simply a sequence of lines containing ASCII characters. It is separated from the headers by a null line (i.e., a line with nothing preceding the CRLF).

Long Header Fields

Each header field can be viewed as a single, logical line of ASCII characters, comprising a field-name and a field-body. For convenience, the field-body portion of this conceptual entity can be split into a multiple-line representation; this is called "folding". The general rule is that wherever there may be linear-white-space (NOT simply LWSP-chars), a CRLF immediately followed by AT LEAST one LWSP-char may instead be inserted. Thus, the single line

```
To: "Joe & J.  Harvey" <ddd @Org>, JJV @ BBN
```

can be represented as:

```
To: "Joe & J.  Harvey" <ddd @ Org>,
JJV@BBN
```

and

```
To: "Joe & J.  Harvey"
<ddd@ Org>, JJV
@BBN
```

and

```
To: "Joe &
J.  Harvey" <ddd @ Org>, JJV @ BBN
```

The process of moving from this folded multiple-line representation of a header field to its single line representation is called "unfolding". Unfolding is accomplished byregarding CRLF immediately followed by a LWSP-char as equivalent to the LWSP-char.

Note: While the standard permits folding wherever linear-white-space is permitted, it is recommended that structured fields, such as those containing addresses, limit folding to higher-level syntactic breaks. For address fields, it is recommended that such folding occur between addresses, after the separating comma.

Structure Of Header Fields

Once a field has been unfolded, it may be viewed as being composed of a field-name followed by a colon (":"), followed by a field-body, and terminated by a carriage-return/line-feed. The field-name must be composed of printable ASCII characters (i.e., characters that have values between 33. and 126., decimal, except colon). The field-body may be composed of any ASCII characters, except CR or LF. (While CR and/or LF may be present in the actual text, they are removed by the action of unfolding the field.)

Certain field-bodies of headers may be interpreted according to an internal syntax that some systems may wish to parse. These fields are called "structured fields". Examples include fields containing dates and addresses. Other fields, such as "Subject" and "Comments", are regarded simply as strings of text.

Note: Any field which has a field-body that is defined as other than simply <text> is to be treated as a structured field.

Field-names, unstructured field bodies and structured field bodies each are scanned by their own, independent "lexical" analyzers.

Unstructured Field Bodies

For some fields, such as "Subject" and "Comments", no structuring is assumed, and they are treated simply as <text>s, as in the message body. Rules of folding apply to these fields, so that such field bodies which occupy several lines must therefore have the second and successive lines indented by at least one LWSP-char.

Structured Field Bodies

To aid in the creation and reading of structured fields, the free insertion of linear-white-space (which permits folding by inclusion of CRLFs) is allowed between lexical tokens. Rather than obscuring the syntax specifications for these structured fields with explicit syntax for this linear-white-space, the existence of another "lexical" analyzer is assumed. This analyzer does not apply for unstructured field bodies that are simply strings of text, as described above. The analyzer provides an interpretation of the unfolded text composing the body of the field as a sequence of lexical symbols.

These symbols are:

- individual special characters
- quoted-strings
- domain-literals
- comments
- atoms

The first four of these symbols are self-delimiting. Atoms are not; they are delimited by the self-delimiting symbols and by linear-white-space. For the purposes of regenerating sequences of atoms and quoted-strings, exactly one SPACE is assumed to exist, and should be used, between them. (Also, in the "Clarifications" section on "White Space", below, note the rules about treatment of multiple contiguous LWSP-chars.)

So, for example, the folded body of an address field

```
":sysmail"@ Some-Group.  Some-Org,
Muhammed.(I am the greatest) Ali @(the)Vegas.WBA
```

is analyzed into the following lexical symbols and types:

```
:sysmail                quoted string
@                       special
Some-Group              atom
.                       special
Some-Org                atom
,                       special
Muhammed                atom
.                       special
(I am the greatest)     comment
Ali                     atom
@                       atom
(the)                   comment
Vegas                   atom
.                       special
WBA                     atom
```

The canonical representations for the data in these addresses are the following strings:

```
":sysmail"@Some-Group.Some-Org
```

and

```
Muhammed.Ali@Vegas.WBA
```

Note: For purposes of display, and when passing such structured information to other systems, such as mail protocol services, there must be NO linear-white-space between <word>s that are separated by period (".") or at-sign ("@") and exactly one SPACE between all other <word>s. Also, headers should be in a folded form.

Header Field Definitions

These show a field meta-syntax, without regard for the particular type or internal syntax. Their purpose is to permit detection of fields; also, they present to higher-level parsers an image of each field as fitting on one line.

```
field    = field-name ":" [ field-body ] CRLF

field-name = 1*<any CHAR, excluding CTLs, SPACE, and ":">

field-body = field-body-contents
[CRLF LWSP-char field-body]

field-body-contents =<the ASCII characters making up the field-body, as
defined in the following sections, and consisting of combinations of
atom, quoted-string, and specials tokens, or else consisting of texts>
```

Lexical Tokens

The following rules are used to define an underlying lexical analyzer, which feeds tokens to higher level parsers. See the ANSI bibliographic references.

```
                    ; ( Octal, Decimal.)
CHAR    = <any ASCII character>    ; ( 0-177, 0.-127.)
ALPHA    = <any ASCII alphabetic character>
                    ; (101-132, 65.- 90.)
                    ; (141-172, 97.-122.)
DIGIT    = <any ASCII decimal digit>  ; ( 60- 71, 48.- 57.)
CTL     = <any ASCII control      ; ( 0- 37, 0.- 31.)
        character and DEL>     ; (  177,   127.)
CR     = <ASCII CR, carriage return> ; (   15,   13.)
LF     = <ASCII LF, linefeed>     ; (   12,   10.)
SPACE    = <ASCII SP, space>      ; (   40,   32.)
HTAB    = <ASCII HT, horizontal-tab>  ; (   11,    9.)
<">     = <ASCII quote mark>      ; (   42,   34.)
CRLF    = CR LF

LWSP-char  = SPACE / HTAB         ; semantics = SPACE

linear-white-space = 1*([CRLF] LWSP-char) ; semantics = SPACE
                    ; CRLF => folding

specials  = "(" / ")" / "<" / ">" / "@" ; Must be in quoted-
      / "," / ";" / ":" / "\" / <"> ; string, to use
      / "." / "[" / "]"       ; within a word.

delimiters = specials / linear-white-space / comment

text    = <any CHAR, including bare  ; => atoms, specials,
        CR & bare LF, but NOT    ; comments and
        including CRLF>       ; quoted-strings are
                    ; NOT recognized.

atom    = 1*<any CHAR except specials, SPACE and CTLs>

quoted-string = <"> *(qtext/quoted-pair) <">; Regular qtext or
                    ;  quoted chars.

qtext    = <any CHAR excepting <">,   ; => may be folded
        "\" & CR, and including
        linear-white-space>

domain-literal = "[" *(dtext / quoted-pair) "]"

dtext    = <any CHAR excluding "[",   ; => may be folded
        "]", "\" & CR, & including
        linear-white-space>

comment   = "(" *(ctext / quoted-pair / comment) ")"

ctext    = <any CHAR excluding "(",   ; => may be folded
```

```
        ")", "\" & CR, & including
        linear-white-space>

quoted-pair = "\" CHAR           ; may quote any char

phrase   = 1*word            ; Sequence of words

word    = atom / quoted-string
```

Clarifications

Quoting

Some characters are reserved for special interpretation, such as delimiting lexical tokens. To permit use of these characters as uninterpreted data, a quoting mechanism is provided. To quote a character, precede it with a backslash ("\").

This mechanism is not fully general. Characters may be quoted only within a subset of the lexical constructs. In particular, quoting is limited to use within:

- quoted-string
- domain-literal
- comment

Within these constructs, quoting is **REQUIRED** for CR and "\" and for the character(s) that delimit the token (e.g., "(" and ")" for a comment). However, quoting is **PERMITTED** for any character.

Note: In particular, quoting is **NOT** permitted within atoms. For example when the local-part of an addr-spec must contain a special character, a quoted string must be used. Therefore, a specification such as:

```
Full\ Name@Domain
```

is not legal and must be specified as:

```
"Full Name"@Domain
```

White Space

Note: In structured field bodies, multiple linear space ASCII characters (namely HTABs and SPACEs) are treated as single spaces and may freely surround any symbol. In all header fields, the only place in which at least one LWSP-char is **REQUIRED** is at the beginning of continua-
tion lines in a folded field.

When passing text to processes that do not interpret text according to this standard (e.g., mail protocol servers), then **NO** linear-white-space characters should occur between a period (".") or at-sign ("@") and a <word>. Exactly **ONE SPACE** should be used in place of arbitrary linear-white-space and comment sequences.

Note: Within systems conforming to this standard, wherever a member of the list of delimiters is allowed, LWSP-chars may also occur before and/or after it.

Writers of mail-sending (i.e., header-generating) programs should realize that there is no network-wide definition of the effect of ASCII HT (horizontal-tab) characters on the appearance of text at another network host; therefore, the use of tabs in message headers, though permitted, is discouraged.

Comments

A comment is a set of ASCII characters, which is enclosed in matching parentheses and which is not within a quoted-string The comment construct permits message originators to add text which will be useful for human readers, but which will be ignored by the formal semantics. Comments should be retained while the message is subject to interpretation according to this standard. However, comments must NOT be included in other cases, such as during protocol exchanges with mail servers.

Comments nest, so that if an unquoted left parenthesis occurs in a comment string, there must also be a matching right parenthesis. When a comment acts as the delimiter between a sequence of two lexical symbols, such as two atoms, it is lexically equivalent with a single **SPACE**, for the purposes of regenerating the sequence, such as when passing the sequence onto a mail protocol server. Comments are detected as such only within field-bodies of structured fields.

If a comment is to be "folded" onto multiple lines, then the syntax for folding must be adhered to. (See the "Lexical Analysis of Messages" section on "Folding Long Header Fields", and the section on "Case Independence" below.) Note that the official semantics therefore do not "see" any unquoted CRLFs that are in comments, although particular parsing programs may wish to note their presence. For these programs, it would be reasonable to interpret a "CRLF LWSP-char" as being a CRLF that is part of the comment; i.e., the CRLF is kept and the LWSP-char is discarded. Quoted CRLFs (i.e., a backslash followed by a CR followed by a LF) still must be followed by at least one LWSP-char.

Delimiting And Quoting Characters

The quote character (backslash) and characters that delimit syntactic units are not, generally, to be taken as data that are part of the delimited or quoted unit(s). In particular, the quotation-marks that define a quoted-string, the parentheses that define a comment and the backslash that quotes a following character are **NOT** part of the quoted-string, comment or quoted character. A quotation-mark that is to be part of a quoted-string, a parenthesis that is to be part of a comment and a backslash that is to be part of either must each be preceded by the quote-character backslash ("\"). Note that the syntax allows any character to be quoted within a quoted-string or comment; however only certain characters **MUST** be quoted to be included as data. These characters are the ones that are not part of the alternate text group (i.e., ctext or qtext).

The one exception to this rule is that a single **SPACE** is assumed to exist between contiguous words in a phrase, and this interpretation is independent of the actual number of LWSP-chars that the creator places between the words. To include more than one **SPACE**, the creator must make the LWSP-chars be part of a quoted-string.

Quotation marks that delimit a quoted string and backslashes that quote the following character should **NOT** accompany the quoted-string when the string is passed to processes that do not interpret data according to this specification (e.g., mail protocol servers).

Quoted-Strings

Where permitted (i.e., in words in structured fields) quoted-strings are treated as a single symbol. That is, a quoted- string is equivalent to an atom, syntactically. If a quoted-string is to be "folded" onto multiple lines, then the syntax for folding must be adhered to. (See the "Lexical Analysis of Messages" section on "Folding Long Header Fields", and the section on "Case Independence" below.)

Therefore, the official semantics do not "see" any bare CRLFs that are in quoted-strings; however particular parsing programs may wish to note their presence. For such programs, it would be reasonable to interpret a "CRLF LWSP-char" as being a CRLF which is part of the quoted-string; i.e., the CRLF is kept and the LWSP-char is discarded. Quoted CRLFs (i.e., a backslash followed by a CR followed by a LF) are also subject to rules of folding, but the presence of the quoting character (backslash) explicitly indicates that the CRLF is data to the quoted string. Stripping off the first following LWSP-char is also appropriate when parsing quoted CRLFs.

Bracketing Characters

There is one type of bracket which must occur in matched pairs and may have pairs nested within each other:

- Parentheses ("(" and ")") are used to indicate comments.

There are three types of brackets which must occur in matched pairs, and which may **NOT** be nested:

- Colon/semi-colon (":" and ";") are used in address specifications to indicate that the included list of addresses are to be treated as a group.

- Angle brackets ("<" and ">") are generally used to indicate the presence of a one machine-usable reference (e.g., delimiting mailboxes), possibly including source-routing to the machine.

- Square brackets ("[" and "]") are used to indicate the presence of a domain-literal, which the appropriate name-domain is to use directly, bypassing normal name-resolution mechanisms.

Case Independence

Except as noted, alphabetic strings may be represented in any combination of upper and lower case. The only syntactic units which requires preservation of case information are:

- text
- qtext
- dtext
- ctext
- quoted-pair
- local-part, except "Postmaster"

When matching any other syntactic unit, case is to be ignored. For example, the field-names "From", "FROM", "from", and even "FroM" are semantically equal and should all be treated identically.

When generating these units, any mix of upper and lower case alphabetic characters may be used. The case shown in this specification is suggested for message-creating processes.

Note: The reserved local-part address unit, "Postmaster", is an exception. When the value "Postmaster" is being interpreted, it must be accepted in any mixture of case, including "POSTMASTER", and "postmaster".

Folding Long Header Fields

Each header field may be represented on exactly one line consisting of the name of the field and its body, and terminated by a CRLF; this is what the parser sees. For readability, the field-body portion of long header fields may be "folded" onto multiple lines of the actual field. "Long" is commonly interpreted to mean greater than 65 or 72 characters. The former length serves as a limit, when the message is to be viewed on most simple terminals which use simple display software; however, the limit is not imposed by this standard.

Note: Some display software often can selectively fold lines, to suit the display terminal. In such cases, sender-provided folding can interfere with the display software.

Backspace Characters

ASCII BS characters (Backspace, decimal 8) may be included in texts and quoted-strings to effect overstriking. However, any use of backspaces which effects an overstrike to the left of the beginning of the text or quoted-string is prohibited.

Network-Specific Transformations

During transmission through heterogeneous networks, it may be necessary to force data to conform to a network's local conventions. For example, it may be required that a CR be followed either by LF, making a CRLF, or by <null>, if the CR is to stand alone). Such transformations are reversed, when the message exits that network.

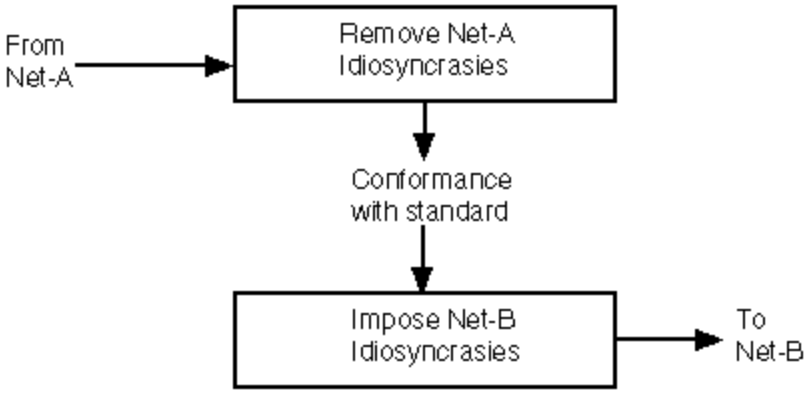
When crossing network boundaries, the message should be treated as passing through two modules. It will enter the first module containing whatever network-specific transformations that were necessary to permit migration through the current" network. It then passes through the modules:

- Transformation Reversal

The "current" network's idiosyncracies are removed and the message is returned to the canonical form specified in this standard.

- Transformation

The "next" network's local idiosyncracies are imposed on the message.



Message Specification

RFC-822 Standard for ARPA Internet Text Messages: Message Specification

Syntax

Note: Due to an artifact of the notational conventions, the syntax indicates that, when present, some fields, must be in a particular order. Header fields are NOT required to occur in any particular order, except that the message body must occur AFTER the headers. It is recommended that, if present, headers be sent in the order "Return-Path", "Received", "Date", "From", "Subject", "Sender", "To", "cc", etc.

This specification permits multiple occurrences of most fields. Except as noted, their interpretation is not specified here, and their use is discouraged.

The following syntax for the bodies of various fields should be thought of as describing each field body as a single long string (or line). The "Lexical Analysis of Message" section on "Long Header Fields" indicates how such long strings can be represented on more than one line in the actual transmitted message.

```
message = fields *( CRLF *text ) ; Everything after
   ; first null line
   ; is message body

fields = dates  ; Creation time,
 source  ; author id & one
 1*destination ; address required
 *optional-field  ; others optional

source = [ trace ]   ; net traversals
  originator   ; original mail
  [ resent ]   ; forwarded

trace = return   ; path to sender
  1*received   ; receipt tags

return = "Return-path" ":" route-addr ; return address
  / "Return-path" ":" "<" ">"

received = "Received" ":"  ; one per relay
  ["from" domain]  ; sending host
  ["by" domain]  ; receiving host
  ["via" atom]  ; physical path
  *("with" atom)  ; link/mail protocol
  ["id" msg-id]  ; receiver msg id
  ["for" addr-spec] ; initial form
  ";" date-time  ; time received

originator = authentic   ; authenticated addr
  [ "Reply-To" ":" 1#address] )

authentic = "From" ":" mailbox ; Single author
  / ( "Sender" ":" mailbox ; Actual submittor
  "From" ":" 1#mailbox) ; Multiple authors
      ; or not sender

resent = resent-authentic
```

```
  [ "Resent-Reply-To" ":" 1#address] )

resent-authentic =
  = "Resent-From" ":" mailbox
  / ( "Resent-Sender" ":" mailbox
  "Resent-From" ":" 1#mailbox )

dates = orig-date   ; Original
  [ resent-date ]  ; Forwarded

orig-date = "Date" ":" date-time

resent-date = "Resent-Date" ":" date-time

destination = "To"  ":" 1#address ; Primary
  / "Resent-To" ":" 1#address
  / "cc"  ":" 1#address ; Secondary
  / "Resent-cc" ":" 1#address
  / "bcc"  ":" #address ; Blind carbon
  / "Resent-bcc" ":" #address

optional-field =
  / "Message-ID" ":" msg-id
  / "Resent-Message-ID" ":" msg-id
  / "In-Reply-To" ":" *(phrase / msg-id)
  / "References" ":" *(phrase / msg-id)
  / "Keywords"  ":" #phrase
  / "Subject"  ":" *text
  / "Comments"  ":" *text
  / "Encrypted"  ":" 1#2word
  / "Content-Type" ":" = type [";" ver-num [";" 1#resource-ref]] [comment]
     (for details see RFC-1049)
  / extension-field  ; To be defined
  / user-defined-field  ; May be pre-empted

msg-id = "<" addr-spec ">"  ; Unique message id

extension-field =
  <Any field which is defined in a document
  published as a formal extension to this
  specification; none will have names beginning
  with the string "X-">

user-defined-field =
  <Any field which has not been defined
  in this specification or published as an
  extension to this specification; names for
  such fields must be unique and may be
  pre-empted by published extensions>
```

FORWARDING

Some systems permit mail recipients to forward a message, retaining the original headers, by adding some new fields. This standard supports such a service, through the "Resent-" prefix to field names.

Whenever the string "Resent-" begins a field name, the field has the same semantics as a field whose name does not have the prefix. However, the message is assumed to have been forwarded by an original recipient who attached the "Resent-" field. This new field is treated as being more recent than the equivalent, original field. For example, the "Resent-From", indicates the person that forwarded the message, whereas the "From" field indicates the original author.

Use of such precedence information depends upon participants' communication needs. For example, this standard does not dictate when a "Resent-From:" address should receive replies, in lieu of sending them to the "From:" address.

Note: In general, the "Resent-" fields should be treated as containing a set of information that is independent of the set of original fields. Information for one set should not automatically be taken from the other. The interpretation of multiple "Resent-" fields, of the same type, is undefined.

In the remainder of this specification, occurrence of legal "Resent-" fields are treated identically with the occurrence of fields whose names do not contain this prefix.

Trace Fields

Trace information is used to provide an audit trail of message handling. In addition, it indicates a route back to the sender of the message.

The list of known "via" and "with" values are registered with the Network Information Center, SRI International, Menlo Park, California.

Return Path
Received

Return-Path

This field is added by the final transport system that delivers the message to its recipient. The field is intended to contain definitive information about the address and route back to the message's originator.

Note: The "Reply-To" field is added by the originator and serves to direct replies, whereas the "Return-Path" field is used to identify a path back to the originator.

While the syntax indicates that a route specification is optional, every attempt should be made to provide that information in this field.

Received

A copy of this field is added by each transport service that relays the message. The information in the field can be quite useful for tracing transport problems.

The names of the sending and receiving hosts and time-of-receipt may be specified. The "via" parameter may be used, to indicate what physical mechanism the message was sent over, such as Arpanet or Phonenet, and the "with" parameter may be used to indicate the mail-, or connection-, level protocol that was used, such as the SMTP mail protocol, or X.25 transport protocol.

Note: Several "with" parameters may be included, to fully specify the set of protocols that were used.

Some transport services queue mail; the internal message identifier that is assigned to the message may be noted, using the "id" parameter. When the sending host uses a destination address specification that the receiving host reinterprets, by expansion or transformation, the receiving host may wish to record the original specification, using the "for" parameter. For example, when a copy of mail is sent to the member of a distribution list, this parameter may be used to record the original address that was used to specify the list.

Originator Fields

The standard allows only a subset of the combinations possible with the From, Sender, Reply-To, Resent-From, Resent-Sender, and Resent-Reply-To fields. The limitation is intentional.

From / Resent-From

This field contains the identity of the person(s) who wished this message to be sent. The message-creation process should default this field to be a single, authenticated machine address, indicating the **AGENT** (person, system or process) entering the message. If this is not done, the "Sender" field **MUST** be present. If the "From" field IS defaulted this way, the "Sender" field is optional and is redundant with the "From" field. In all cases, addresses in the "From" field must be machine-usable (addr-specs) and may not contain named lists (groups).

RFC-822 Standard for ARPA Internet Text Messages: Message Specification

Sender / Resent-Sender

This field contains the authenticated identity of the **AGENT** (person, system or process) that sends the message. It is intended for use when the sender is not the author of the message, or to indicate who among a group of authors actually sent the message. If the contents of the "Sender" field would be completely redundant with the "From" field, then the "Sender" field need not be present and its use is discouraged (though still legal). In particular, the "Sender" field **MUST** be present if it is NOT the same as the "From" Field.

The Sender mailbox specification includes a word sequence which must correspond to a specific agent (i.e., a human user or a computer program) rather than a standard address. This indicates the expectation that the field will identify the single **AGENT** (person, system, or process) responsible for sending the mail and not simply include the name of a mailbox from which the mail was sent. For example in the case of a shared login name, the name, by itself, would not be adequate. The local-part address unit, which refers to this agent, is expected to be a computer system term, and not (for example) a generalized person reference which can be used outside the network text message context.

Since the critical function served by the "Sender" field is identification of the agent responsible for sending mail and since computer programs cannot be held accountable for their behavior, it is strongly recommended that when a computer program generates a message, the **HUMAN** who is responsible for that program be referenced as part of the "Sender" field mailbox specification.

Reply-To / Resent-Reply-To

This field provides a general mechanism for indicating any mailbox(es) to which responses are to be sent. Three typical uses for this feature can be distinguished. In the first case, the author(s) may not have regular machine-based mailboxes and therefore wish(es) to indicate an alternate machine address. In the second case, an author may wish additional persons to be made aware of, or responsible for, replies. A somewhat different use may be of some help to "text message teleconferencing" groups equipped with automatic distribution services: include the address of that service in the "Reply-To" field of all messages submitted to the teleconference; then participants can "reply" to conference submissions to guarantee the correct distribution of any submission of their own.

Note: The "Return-Path" field is added by the mail transport service, at the time of final deliver. It is intended to identify a path back to the orginator of the message. The "Reply-To" field is added by the message originator and is intended to direct replies.

Automatic Use Of From / Sender / Reply-To

For systems which automatically generate address lists for replies to messages, the following recommendations are made:

- The "Sender" field mailbox should be sent notices of any problems in transport or delivery of the original messages. If there is no "Sender" field, then the "From" field mailbox should be used.

- The "Sender" field mailbox should **NEVER** be used automatically, in a recipient's reply message.

- If the "Reply-To" field exists, then the reply should go to the addresses indicated in that field and not to the address(es) indicated in the "From" field.

- If there is a "From" field, but no "Reply-To" field, the reply should be sent to the address(es) indicated in the "From" field.

Sometimes, a recipient may actually wish to communicate with the person that initiated the message transfer. In such cases, it is reasonable to use the "Sender" address.

This recommendation is intended only for automated use of originator-fields and is not intended to suggest that replies may not also be sent to other recipients of messages. It is up to the respective mail-handling programs to decide what additional facilities will be provided.

Examples are provided in Appendix A.

Receiver Fields

To / Resent-To
CC / Resent-CC
BCC / Resent-BCC

To / Resent-To

This field contains the identity of the primary recipients of the message.

CC / Resent-CC

This field contains the identity of the secondary (informational) recipients of the message.

BCC / Resent-BCC

This field contains the identity of additional recipients of the message. The contents of this field are not included in copies of the message sent to the primary and secondary recipients. Some systems may choose to include the text of the "BCC" field only in the author(s)'s copy, while others may also include it in the text sent to all those indicated in the "BCC" list.

Reference Fields

Message-ID / Resent-Message-ID
In-Reply-To
References
Keywords

Message-ID / Resent-Message-ID

This field contains a unique identifier (the local-part address unit) which refers to **THIS** version of **THIS** message. The uniqueness of the message identifier is guaranteed by the host which generates it. This identifier is intended to be machine readable and not necessarily meaningful to humans. A message identifier pertains to exactly one instantiation of a particular message; subsequent revisions to the message should each receive new message identifiers.

In-Reply-To

The contents of this field identify previous correspondence which this message answers. Note that if message identifiers are used in this field, they must use the msg-id specification format.

References

The contents of this field identify other correspondence which this message references. Note that if message identifiers are used, they must use the msg-id specification format.

Keywords

This field contains keywords or phrases, separated by commas.

Other Fields

Subject

This is intended to provide a summary, or indicate the nature, of the message.

Comments

Permits adding text comments onto the message without disturbing the contents of the message's body.

Encrypted

Sometimes, data encryption is used to increase the privacy of message contents. If the body of a message has been encrypted, to keep its contents private, the "Encrypted" field can be used to note the fact and to indicate the nature of the encryption. The first <word> parameter indicates the software used to encrypt the body, and the second, optional <word> is intended to aid the recipient in selecting the proper decryption key. This code word may be viewed as an index to a table of keys held by the recipient.

Note: Unfortunately, headers must contain envelope, as well as contents, information. Consequently, it is necessary that they remain unencrypted, so that mail transport services may access them. Since names, addresses, and "Subject" field contents may contain sensitive information, this requirement limits total message privacy.

Names of encryption software are registered with the Network Information Center, SRI International, Menlo Park, California.

Extension-Field

A limited number of common fields have been defined in this document. As network mail requirements dictate, additional fields may be standardized. To provide user-defined fields with a measure of safety, in name selection, such extension-fields will never have names that begin with the string "X-".

Names of Extension-fields are registered with the Network Information Center, SRI International, Menlo Park, California.

User-Defined-Field

Individual users of network mail are free to define and use additional header fields. Such fields must have names which are not already used in the current specification or in any definitions of extension-fields, and the overall syntax of these user-defined-fields must conform to this specification's rules for delimiting and folding fields. Due to the extension-field publishing process, the name of a user-defined-field may be pre-empted

Note: The prefatory string "X-" will never be used in the names of Extension-fields. This provides user-defined fields with a protected set of names.

RFC-822 Standard for ARPA Internet Text Messages: Date and Time

Syntax

All mail software **should** use 4-digit years in dates, to ease the transition to the next century.

There is a strong trend towards the use of numeric time zone indicators, and implementations **should** use numeric timezones instead of timezone names. However, all implementations **must** accept either notation. If timezone names are used, they **must** be exactly as defined below.

NOTE: The military time zones are specified incorrectly in RFC-822: they count the wrong way from UT (the signs are reversed). As a result, military time zones in RFC-822 headers carry no information.

```
date-time = [ day "," ] date time ; dd mm yy
      ; hh:mm:ss zzz

day  = "Mon" / "Tue" / "Wed" / "Thu"
  / "Fri" / "Sat" / "Sun"

date = 1*2DIGIT month 2*4DIGIT ; day month year
      ; e.g.  20 Jun 82

month = "Jan" / "Feb" / "Mar" / "Apr"
  / "May" / "Jun" / "Jul" / "Aug"
  / "Sep" / "Oct" / "Nov" / "Dec"

time = hour zone   ; ANSI and Military

hour = 2DIGIT ":" 2DIGIT [":" 2DIGIT]
      ; 00:00:00 - 23:59:59

zone = "UT" / "GMT"  ; Universal Time
      ; North American : UT
  / "EST" / "EDT"  ; Eastern: - 5/ - 4
  / "CST" / "CDT"  ; Central: - 6/ - 5
  / "MST" / "MDT"  ; Mountain: - 7/ - 6
  / "PST" / "PDT"  ; Pacific: - 8/ - 7
  / 1ALPHA   ; Military: Z = UT;
      ; A:-1; (J not used)
      ; M:-12; N:+1; Y:+12
  / ( ("+" / "-") 4DIGIT ) ; Local differential
      ; hours+min.  (HHMM)
```

Semantics

If included, day-of-week must be the day implied by the date specification.

Time zone may be indicated in several ways. "UT" is Universal Time (formerly called "Greenwich Mean Time"); "GMT" is permitted as a reference to Universal Time. The military standard uses a single character for each zone. "Z" is Universal Time. "A" indicates one hour earlier, and "M" indicates 12 hours earlier; "N" is one hour later, and "Y" is 12 hours later. The letter "J" is not used. The other remaining two forms are taken from ANSI standard X3.51-1975. One allows explicit indication of the amount of offset from UT; the other uses common 3-characterstrings for indicating time zones in North America.

Address Specification

Syntax

```
address = mailbox   ; one addressee
  / group   ; named list

group = phrase ":" [#mailbox] ";"

mailbox = addr-spec   ; simple address
  / [phrase] route-addr  ; name & addr-spec

route-addr = "<" [route] addr-spec ">"

route = 1#("@" domain) ":"  ; path-relative

addr-spec = local-part "@" domain ; global address

local-part = word *("." word)  ; uninterpreted
      ; case-preserved

domain = sub-domain *("." sub-domain)

sub-domain = domain-ref / domain-literal

domain-ref = atom    ; symbolic reference
```

Semantics

A mailbox receives mail. It is a conceptual entity which does not necessarily pertain to file storage. For example, some sites may choose to print mail on their line printer and deliver the output to the addressee's desk.

A mailbox specification comprises a person, system or process name reference, a domain-dependent string, and a name-domain reference. The name reference is optional and is usually used to indicate the human name of a recipient. The name-domain reference specifies a sequence of sub-domains. The domain-dependent string is uninterpreted, except by the final sub-domain; the rest of the mail service merely transmits it as a literal string.

The basic mailbox address specification has the form: "`local-part@domain`". Here "`local-part`", sometimes called the "left- hand side" of the address, is domain-dependent.

A host that is forwarding the message but is not the destination host implied by the right-hand side "domain" **must not** interpret or modify the "`local-part`" of the address.

When mail is to be gatewayed from the Internet mail environment into a foreign mail environment, routing information for that foreign environment **may** be embedded within the "local-part" of the address. The gateway will then interpret this local part appropriately for the foreign mail environment. See Discussion section for further information.

Discussion

Although source routes are discouraged within the Internet (see Mail Relay), there are non-Internet mail environments whose delivery mechanisms do depend upon source routes. Source routes for extra-Internet environments can generally be buried in the "`local-part`" of the address (see Balancing Local-Part And Domain) while mail traverses the Internet. When the mail reaches the appropriate Internet mail gateway, the gateway will interpret the local-part and build the necessary address or route for the target mail environment.

For example, an Internet host might send mail to: "`a!b!c!user@gateway-domain`". The complex local part "`a!b!c!user`" would be uninterpreted within the Internet domain, but could be parsed and understood by the specified mail gateway.

An embedded source route is sometimes encoded in the "`local-part`" using "%" as a right-binding routing operator. For example, in:

```
user%domain%relay3%relay2@relay1
```

the "`%`" convention implies that the mail is to be routed from "`relay1`" through "`relay2`", "`relay3`", and finally to "`user`" at "`domain`". This is commonly known as the "`%- hack`". It is suggested that "`%`" have lower precedence than any other routing operator (e.g., "`!`") hidden in the local-part; for example, "`a!b%c`" would be interpreted as "`(a!b)%c`".

Only the target host (in this case, "`relay1`") is permitted to analyze the local-part "`user%domain%relay3%relay2`".

Domains

A name-domain is a set of registered (mail) names. A name-domain specification resolves to a subordinate name-domain specification or to a terminal domain-dependent string. Hence, domain specification is extensible, permitting any number of registration levels.

Name-domains model a global, logical, hierarchical addressing scheme. The model is logical, in that an address specification is related to name registration and is not necessarily tied to transmission path. The model's hierarchy is a directed graph, called an in-tree, such that there is a single path from the root of the tree to any node in the hierarchy. If more than one path actually exists, they are considered to be different addresses.

The root node is common to all addresses; consequently, it is not referenced. Its children constitute "top-level" name-domains. Usually, a service has access to its own full domain specification and to the names of all top-level name-domains.

The "top" of the domain addressing hierarchy -- a child of the root -- is indicated by the right-most field, in a domain specification. Its child is specified to the left, its child to the left, and so on.

Some groups provide formal registration services; these constitute name-domains that are independent logically of specific machines. In addition, networks and machines implicitly compose name-domains, since their membership usually is registered in name tables.

In the case of formal registration, an organization implements a (distributed) data base which provides an address-to-route mapping service for addresses of the form:

```
person@registry.organization
```

Note that "organization" is a logical entity, separate from any particular communication network.

A mechanism for accessing "organization" is universally available. That mechanism, in turn, seeks an instantiation of the registry; its location is not indicated in the address specification. It is assumed that the system which operates under the name "organization" knows how to find a subordinate registry. The registry will then use the "person" string to determine where to send the mail specification.

The latter, network-oriented case permits simple, direct, attachment-related address specification, such as:

```
user@host.network
```

Once the network is accessed, it is expected that a message will go directly to the host and that the host will resolve the user name, placing the message in the user's mailbox.

Abbreviated Domain Specification

Since any number of levels is possible within the domain hierarchy, specification of a fully qualified address can become inconvenient. This standard permits abbreviated domain specification, in a special case:

For the address of the sender, call the left-most sub-domain Level N. In a header address, if all of the sub-domains above (i.e., to the right of) Level N are the same as those of the sender, then they do not have to appear in the specification. Otherwise, the address must be fully qualified.

This feature is subject to approval by local sub-domains. Individual sub-domains may require their member systems, which originate mail, to provide full domain specification only. When permitted, abbreviations may be present only while the message stays within the sub-domain of the sender.

Use of this mechanism requires the sender's sub-domain to reserve the names of all top-level domains, so that full specifications can be distinguished from abbreviated specifications.

For example, if a sender's address is:

```
sender@registry-A.registry-1.organization-X
```

and one recipient's address is:

```
recipient@registry-B.registry-1.organization-X
```

and another's is:

```
recipient@registry-C.registry-2.organization-X
```

then "`.registry-1.organization-X`" need not be specified in the the message, but "`registry-C.registry-2`" **DOES** have to be specified. That is, the first two addresses may be abbreviated, but the third address must be fully specified.

When a message crosses a domain boundary, all addresses must be specified in the full format, ending with the top-level name-domain in the right-most field. It is the responsibility of mail forwarding services to ensure that addresses conform with this requirement. In the case of abbreviated addresses, the relaying service must make the necessary expansions. It should be noted that it often is difficult for such a service to locate all occurrences of address abbreviations. For example, it will not be possible to find such abbreviations within the body of the message. The "Return-Path" field can aid recipients in recovering from these errors.

Note: When passing any portion of an addr-spec onto a process which does not interpret data according to this standard (e.g., mail protocol servers). There must be **NO** LWSP-chars preceding or following the at-sign or any delimiting period ("."), such as shown in the above examples, and only **ONE SPACE** between contiguous <word>s.

Domain Terms

A domain-ref must be **THE** official name of a registry, network, or host. It is a symbolic reference, within a name sub-domain. At times, it is necessary to bypass standard mechanisms for resolving such references, using more primitive information, such as a network host address rather than its associated host name.

To permit such references, this standard provides the domain-literal construct. Its contents must conform with the needs of the sub-domain in which it is interpreted.

A mailer **must** be able to accept and parse an Internet domain literal whose content ("dtext" in Lexical Tokens) is a dotted- decimal host address. This satisfies the requirement of Rule Naming under Notational Conventions for the case of mail.

An SMTP **must** accept and recognize a domain literal for any of its own IP addresses.

Domain-literals which refer to domains within the ARPA Internet specify 32-bit Internet addresses, in four 8-bit fields noted in decimal, as described in RFC-820, "Assigned Numbers." For example:

```
[10.0.3.19]
```

Note: THE USE OF DOMAIN-LITERALS IS STRONGLY DISCOURAGED. It is permitted only as a means of bypassing temporary system limitations, such as name tables which are not complete.

The names of "top-level" domains, and the names of domains under in the ARPA Internet, are registered with the Network Information Center, SRI International, Menlo Park, California.

Domain-Dependent Local String

The local-part of an addr-spec in a mailbox specification (i.e., the host's name for the mailbox) is understood to be whatever the receiving mail protocol server allows. For example, some systems do not understand mailbox references of the form "`P. D. Q. Bach`", but others do.

This specification treats periods (".") as lexical separators. Hence, their presence in local-parts which are not quoted-strings, is detected. However, such occurrences carry **NO** semantics. That is, if a local-part has periods within it, an address parser will divide the local-part into several tokens, but the sequence of tokens will be treated as one uninterpreted unit. The sequence will be re-assembled, when the address is passed outside of the system such as to a mail protocol service.

For example, the address:

```
First.Last@Registry.Org
```

is legal and does not require the local-part to be surrounded with quotation-marks. (However, "First Last" **DOES** require quoting.) The local-part of the address, when passed outside of the mail system, within the Registry.Org domain, is "First.Last", again without quotation marks.

Balancing Local-Part and Domain

In some cases, the boundary between local-part and domain can be flexible. The local-part may be a simple string, which is used for the final determination of the recipient's mailbox. All other levels of reference are, therefore, part of the domain.

For some systems, in the case of abbreviated reference to the local and subordinate sub-domains, it may be possible to specify only one reference within the domain part and place the other, subordinate name-domain references within the local-part. This would appear as:

```
mailbox.sub1.sub2@this-domain
```

Such a specification would be acceptable to address parsers which conform to RFC-733, but do not support this newer Internet standard. While contrary to the intent of this standard, the form is legal.

Also, some sub-domains have a specification syntax which does not conform to this standard. For example:

```
sub-net.mailbox@sub-domain.domain
```

uses a different parsing sequence for local-part than for domain.

Note: As a rule, the domain specification should contain fields which are encoded according to the syntax of this standard and which contain generally-standardized information. The local-part specification should contain only that portion of the address which deviates from the form or intention of the domain field.

Multiple Mailboxes

An individual may have several mailboxes and wish to receive mail at whatever mailbox is convenient for the sender to access. This standard does not provide a means of specifying "any member of" a list of mailboxes.

A set of individuals may wish to receive mail as a single unit (i.e., a distribution list). The <group> construct permits specification of such a list. Recipient mailboxes are specified within the bracketed part (":" – ";"). A copy of the transmitted message is to be sent to each mailbox listed. This standard does not permit recursive specification of groups within groups.

While a list must be named, it is not required that the contents of the list be included. In this case, the <address> serves only as an indication of group distribution and would appear in the form:

```
name:;
```

Some mail services may provide a group-list distribution facility, accepting a single mailbox reference, expanding it to the full distribution list, and relaying the mail to the list's members. This standard provides no additional syntax for indicating such a service. Using the <group> address alternative, while listing one mailbox in it, can mean either that the mailbox reference will be expanded to a list or that there is a group with one member.

Explicit Path Specification

At times, a message originator may wish to indicate the transmission path that a message should follow. This is called source routing. The normal addressing scheme, used in an addr-spec, is carefully separated from such information; the <`route`> portion of a route-addr is provided for such occasions. It specifies the sequence of hosts and/or transmission services that are to be traversed. Both domain-refs and domain-literals may be used.

The use of source routing is discouraged. Unless the sender has special need of path restriction, the choice of transmission route should be left to the mail transport service.

Note: Internet host software **should not** create an RFC-822 header containing an address with an explicit source route, but **must** accept such headers for compatibility with earlier systems.

> **Discussion**
> Many hosts implemented RFC-822 source routes incorrectly, so the syntax cannot be used unambiguously in practice. Many users feel the syntax is ugly. Explicit source routes are not needed in the mail envelope for delivery; see Mail Relaying. For all these reasons, explicit source routes using the RFC-822 notations are not to be used in Internet mail headers.
>
> As stated in Local-part, it is necessary to allow an explicit source route to be buried in the local-part of an address, e.g., using the "%-hack", in order to allow mail to be gatewayed into another environment in which explicit source routing is necessary. The vigilant will observe that there is no way for a User Agent to detect and prevent the use of such implicit source routing when the destination is within the Internet. We can only discourage source routing of any kind within the Internet, as unnecessary and undesirable.

Common Address Formatting Errors

Errors in formatting or parsing 822 addresses are unfortunately common. This section mentions only the most common errors. A User Agent **must** accept all valid RFC-822 address formats, and **must not** generate illegal address syntax.

- **o** A common error is to leave out the semicolon after a group identifier.

- **o** Some systems fail to fully-qualify domain names in messages they generate. The right-hand side of an "@" sign in a header address field **must** be a fully-qualified domain name.

  For example, some systems fail to fully-qualify the From: address; this prevents a "reply" command in the user interface from automatically constructing a return address.

  Discussion
  Although RFC-822 allows the local use of abbreviated domain names within a domain, the application of RFC-822 in Internet mail does not allow this. The intent is that an Internet host must not send an SMTP message header containing an abbreviated domain name in an address field. This allows the address fields of the header to be passed without alteration across the Internet, as required in Mail Relaying.

- o Some systems mis-parse multiple-hop explicit source routes such as:

  `@relay1,@relay2,@relay3:user@domain`.

- o Some systems over-qualify domain names by adding a trailing dot to some or all domain names in addresses or message-ids. This violates RFC-822 syntax.

Reserved Address

It often is necessary to send mail to a site, without knowing any of its valid addresses. For example, there may be mail system dysfunctions, or a user may wish to find out a person's correct address, at that site.

This standard specifies a single, reserved mailbox address (local-part) which is to be valid at each site. Mail sent to that address is to be routed to a person responsible for the site's mail system or to a person with responsibility for general site operation. The name of the reserved local-part address is:

```
Postmaster
```

so that "`Postmaster@domain`" is required to be valid.

Note: This reserved local-part must be matched without sensitivity to alphabetic case, so that "`POSTMASTER`", "`postmaster`", and even "`poStmASteR`" is to be accepted.

Bibliography

ANSI. "USA Standard Code for Information Interchange," X3.4. American National Standards Institute: New York (1968). Also in: Feinler, E. and J. Postel, eds., "ARPANET Protocol Handbook", NIC 7104.

ANSI. "Representations of Universal Time, Local Time Differentials, and United States Time Zone References for Information Interchange," X3.51-1975. American National Standards Institute: New York (1975).

Bemer, R.W., "Time and the Computer." In: Interface Age (Feb.1979).

Bennett, C.J. "JNT Mail Protocol". Joint Network Team, Rutherford and Appleton Laboratory: Didcot, England.

Bhushan, A.K., Pogran, K.T., Tomlinson, R.S., and White, J.E. "Standardizing Network Mail Headers," ARPANET Request for Comments No. 561, Network Information Center No. 18516; SRI International: Menlo Park (September 1973).

Birrell, A.D., Levin, R., Needham, R.M., and Schroeder, M.D. "Grapevine: An Exercise in Distributed Computing," Communications of the ACM 25, 4 (April 1982), 260-274.

Crocker, D.H., Vittal, J.J., Pogran, K.T., Henderson, D.A. "Standard for the Format of ARPA Network Text Message," ARPANET Request for Comments No. 733, Network Information Center No. 41952. SRI International: Menlo Park (November 1977).

Feinler, E.J. and Postel, J.B. ARPANET Protocol Handbook, Network Information Center No. 7104 (NTIS AD A003890). SRI International: Menlo Park (April 1976).

Harary, F. "Graph Theory". Addison-Wesley: Reading, Mass. (1969).

Levin, R. and Schroeder, M. "Transport of Electronic Messages through a Network," TeleInformatics 79, pp. 29-33. North Holland (1979). Also as Xerox Palo Alto Research Center Technical Report CSL-79-4.

Myer, T.H. and Henderson, D.A. "Message Transmission Protocol," ARPANET Request for Comments, No. 680, Network Information Center No. 32116. SRI International: Menlo Park (1975).

NBS. "Specification of Message Format for Computer Based Message Systems, Recommended Federal Information Processing Standard." National Bureau of Standards: Gaithersburg, Maryland (October 1981).

NIC. Internet Protocol Transition Workbook. Network Information Center, SRI-International, Menlo Park, California (March 1982).

Oppen, D.C. and Dalal, Y.K. "The Clearinghouse: A Decentralized Agent for Locating Named Objects in a Distributed Environment," OPD-T8103. Xerox Office Products Division: Palo Alto, CA. (October 1981).

Postel, J.B. "Assigned Numbers," ARPANET Request for Comments, No. 820. SRI

International: Menlo Park (August 1982).

Postel, J.B. "Simple Mail Transfer Protocol," ARPANET Request for Comments, No. 821. SRI International: Menlo Park (August 1982).

Shoch, J.F. "Internetwork naming, addressing and routing," in Proc. 17th IEEE Computer Society International Conference, pp.72-79, Sept. 1978, IEEE Cat. No. 78 CH 1388-8C.

Su, Z. and Postel, J. "The Domain Naming Convention for Internet User Applications," ARPANET Request for Comments, No. 819. SRI International: Menlo Park (August 1982).

Examples

Addresses

1. `Alfred Neuman <Neuman@BBN-TENEXA>`

2. `Neuman@BBN-TENEXA`

These two "Alfred Neuman" examples have identical semantics, as far as the operation of the local host's mail sending (distribution) program (also sometimes called its "mailer") and the remote host's mail protocol server are concerned. In the first example, the "Alfred Neuman" is ignored by the mailer, as "`Neuman@BBN-TENEXA`" completely specifies the recipient. The second example contains no superfluous information, and, again, "`Neuman@BBN-TENEXA`" is the intended recipient.

Note: When the message crosses name-domain boundaries, then these specifications must be changed, so as to indicate the remainder of the hierarchy, starting with the top level.

3. `"George, Ted" <Shared@Group.Arpanet>`

This form might be used to indicate that a single mailbox is shared by several users. The quoted string is ignored by the originating host's mailer, because "`Shared@Group.Arpanet`" completely specifies the destination mailbox.

4. `Wilt . (the Stilt) Chamberlain@NBA.US`

The "(the Stilt)" is a comment, which is **NOT** included in the destination mailbox address handed to the originating system's mailer. The local-part of the address is the string "`Wilt.Chamberlain`", with **NO** space between the first and second words.

5. `Address Lists`

```
Gourmets:   Pompous Person <WhoZiWhatZit@Cordon-Bleu>,
          Childs@WGBH.Boston, Galloping Gourmet@
          ANT.Down-Under (Australian National Television),
          Cheapie@Discount-Liquors;,
Cruisers: Port@Portugal, Jones@SEA;,
     Another@Somewhere.SomeOrg
```

This group list example points out the use of comments and the mixing of addresses and groups.

Originator Items

1. Author-sent

George Jones logs into his host as "Jones". He sends mail himself.

```
From: Jones@Group.Org
```

or

```
From: George Jones <Jones@Group.Org>
```

2. Secretary-sent

George Jones logs in as Jones on his host. His secretary, who logs in as Secy sends mail for him. Replies to the mail should go to George.

```
From: George Jones <Jones@Group>
Sender: Secy@Other-Group
```

3. Secretary-sent, for user of shared directory

George Jones' secretary sends mail for George. Replies should go to George.

```
From:  George Jones<Shared@Group.Org>
Sender: Secy@Other-Group
```

Note that there need not be a space between "Jones" and the "<", but adding a space enhances readability (as is the case in other examples.

4. Committee activity, with one author

George is a member of a committee. He wishes to have any replies to his message go to all committee members.

```
From:  George Jones <Jones@Host.Net>
Sender: Jones@Host
Reply-To: The Committee: Jones@Host.Net,
       Smith@Other.Org,
       Doe@Somewhere-Else;
```

Note that if George had not included himself in the enumeration of The Committee, he would not have gotten an implicit reply; the presence of the "Reply-to" field **SUPERSEDES** the sending of a reply to the person named in the "From" field.

5. Secretary acting as full agent of author

George Jones asks his secretary (`Secy@Host`) to send a message for him in his capacity as Group. He wants his secretary to handle all replies.

```
From:  George Jones <Group@Host>
Sender: Secy@Host
Reply-To: Secy@Host
```

6. Agent for user without online mailbox

A friend of George's, Sarah, is visiting. George's secretary sends some mail to a friend of Sarah in computerland. Replies should go to George, whose mailbox is Jones at Registry.

```
From:   Sarah Friendly <Secy@Registry>
Sender:  Secy-Name <Secy@Registry>
Reply-To: Jones@Registry.
```

7. Agent for member of a committee

George's secretary sends out a message which was authored jointly by all the members of a committee. Note that the name of the committee cannot be specified, since <group> names are not permitted in the From field.

```
From:    Jones@Host,
         Smith@Other-Host,
         Doe@Somewhere-Else
Sender:  Secy@SHost
         Standard for ARPA Internet Text Messages
```

RFC-822 Standard for ARPA Internet Text Messages: Appendix A

Complete Headers

1. Minimum required

```
Date:     26 Aug 76 1429 EDT        Date:     26 Aug 76 1429 EDT
From:     Jones@Registry.Org   or   From:     Jones@Registry.Org
Bcc:                                To:       Smith@Registry.Org
```

Note that the "Bcc" field may be empty, while the "To" field is required to have at least one address.

2. Using some of the additional fields

```
Date:     26 Aug 76 1430 EDT
From:     George Jones<Group@Host>
Sender:   Secy@SHOST
To:       "Al Neuman"@Mad-Host,
          Sam.Irving@Other-Host
Message-ID:  <some.string@SHOST>
```

3. About as complex as you're going to get

```
Date     :  27 Aug 76 0932 PDT
 From     :  Ken Davis <KDavis@This-Host.This-net>
 Subject  :  Re: The Syntax in the RFC
 Sender   :  KSecy@Other-Host
 Reply-To :  Sam.Irving@Reg.Organization
 To       :  George Jones <Group@Some-Reg.An-Org>,
             Al.Neuman@MAD.Publisher
 cc       :  Important folk:
               Tom Softwood <Balsa@Tree.Root>,
               "Sam Irving"@Other-Host;,
             Standard Distribution:
               /main/davis/people/standard@Other-Host,
               "<Jones>standard.dist.3"@Tops-20-Host>;
 Comment  :  Sam is away on business. He asked me to handle
             his mail for him.  He'll be able to provide  a
             more  accurate  explanation  when  he  returns
             next week.
 In-Reply-To: <some.string@DBM.Group>, George's message
 X-Special-action:  This is a sample of user-defined field-
             names.  There could also be a field-name
             "Special-action", but its name might later be
             preempted
 Message-ID: <4231.629.XYzi-What@Other-Host>
```

Simple Field Parsing

Some mail-reading software systems may wish to perform only minimal processing, ignoring the internal syntax of structured field-bodies and treating them the same as unstructured-field-bodies. Such software will need only to distinguish:

- Header fields from the message body,
- Beginnings of fields from lines which continue fields,
- Field-names from field-contents.

The abbreviated set of syntactic rules which follows will suffice for this purpose. It describes a limited view of messages and is a subset of the syntactic rules provided in the main part of this specification. One small exception is that the contents of field-bodies consist only of text:

Syntax

```
message     =  *field *(CRLF *text)

field      =  field-name ":" [field-body] CRLF

field-name   = 1*<any CHAR, excluding CTLs, SPACE, and ":">

field-body   =  *text [CRLF LWSP-char field-body]
```

Semantics

Headers occur before the message body and are terminated by a null line (i.e., two contiguous CRLFs).

A line which continues a header field begins with a SPACE or HTAB character, while a line beginning a field starts with aprintable character which is not a colon.

A field-name consists of one or more printable characters (excluding colon, space, and control-characters). A field-name **MUST** be contained on one line. Upper and lower case are not distinguished when comparing field-names.

Differences From RFC-733

The following summarizes the differences between this standard and the one specified in Arpanet Request for Comments #733 "Standard for the Format of ARPA Network Text Messages". The differences are listed in the order of their occurrence in the current specification.

1. FIELD DEFINITIONS

Field Names

These now must be a sequence of printable characters. They may not contain any LWSP-chars.

2. LEXICAL TOKENS

Specials

The characters period ("."), left-square bracket ("["), and right-square bracket ("]") have been added. For presentation purposes, and when passing a specification to a system that does not conform to this standard, periods are to be contiguous with their surrounding lexical tokens. No linear-white-space is permitted between them. The presence of one LWSP-char between other tokens is still directed.

Atom

Atoms may not contain SPACE.

Special Text

ctext and qtext have had backslash ("\") added to the list of prohibited characters.

Domains

The lexical tokens <domain-literal> and <dtext> have been added.

3. MESSAGE SPECIFICATION

Trace

The "Return-path:" and "Received:" fields have been specified.

From

The "From" field must contain machine-usable addresses (addr-spec). Multiple addresses may be specified, but named-lists (groups) may not.

Resent

The meta-construct of prefacing field names with the string "Resent-" has been added, to indicate that a message has been forwarded by an intermediate recipient.

Destination

A message must contain at least one destination address field. "To" and "CC" are required to contain at least one address.

In-Reply-To

The field-body is no longer a comma-separated list, although a sequence is still permitted.

Reference

The field-body is no longer a comma-separated list, although a sequence is still permitted.

Encrypted

A field has been specified that permits senders to indicate that the body of a message has been encrypted.

Extension-Field

Extension fields are prohibited from beginning with the characters "X-".

4. DATE AND TIME SPECIFICATION

Simplification

Fewer optional forms are permitted and the list of three-letter time zones has been shortened.

5. ADDRESS SPECIFICATION

Address

The use of quoted-string, and the ":"-atom-":" construct, have been removed. An address now is either a single mailbox reference or is a named list of addresses. The latter indicates a group distribution.

Groups

Group lists are now required to to have a name. Group lists may not be nested.

Mailbox

A mailbox specification may indicate a person's name, as before. Such a named list no longer may specify multiple ailboxes and may not be nested.

Route Addressing

Addresses now are taken to be absolute, global specifications, independent of transmission paths. The <route> construct has been provided, to permit explicit specification of transmission path. RFC #733's use of multiple at-signs ("@") was intended as a general syntax for indicating routing and/or hierarchical addressing. The current standard separates these specifications and only one at-sign is

permitted.

At-Sign

The string " at " no longer is used as an address delimiter. Only at-sign ("@") serves the function.

Domains

Hierarchical, logical name-domains have been added.

Reserved Address

The local-part "Postmaster" has been reserved, so that users can be guaranteed at least one valid address at a site.

Alphabetical Listing Of Syntax Rules

```
ALPHA       =  <any ASCII alphabetic character>
                                          ; (101-132, 65.- 90.)
                                          ; (141-172, 97.-122.)
atom        =  1*<any CHAR except specials, SPACE and CTLs>
authentic   =   "From"       ":"   mailbox  ; Single author
            / ( "Sender"     ":"   mailbox  ; Actual submittor
                "From"       ":" 1#mailbox) ; Multiple authors
                                          ;  or not sender
CHAR        =  <any ASCII character>        ; (  0-177,  0.-127.)
comment     =  "(" *(ctext / quoted-pair / comment) ")"
CR          =  <ASCII CR, carriage return>  ; (     15,      13.)
CRLF        =  CR LF
ctext       =  <any CHAR excluding "(",     ; => may be folded
                ")", "\" & CR, & including
                linear-white-space>
CTL         =  <any ASCII control           ; (  0- 37,  0.- 31.)
                character and DEL>          ; (    177,     127.)
date        =  1*2DIGIT month 2DIGIT        ; day month year
                                          ;  e.g. 20 Jun 82
dates       =   orig-date                   ; Original
              [ resent-date ]               ; Forwarded
date-time   =  [ day "," ] date time        ; dd mm yy
                                          ;  hh:mm:ss zzz
day         =  "Mon"  / "Tue" /  "Wed"  / "Thu"
            /  "Fri"  / "Sat" /  "Sun"
delimiters  =  specials / linear-white-space / comment
destination =  "To"           ":" 1#address  ; Primary
            /  "Resent-To"    ":" 1#address
            /  "cc"           ":" 1#address  ; Secondary
            /  "Resent-cc"    ":" 1#address
            /  "bcc"          ":"  #address  ; Blind carbon
            /  "Resent-bcc"   ":"  #address
DIGIT       =  <any ASCII decimal digit>    ; ( 60- 71, 48.- 57.)
domain      =  sub-domain *("." sub-domain)
domain-literal =  "[" *(dtext / quoted-pair) "]"
domain-ref  =  atom                         ; symbolic reference
dtext       =  <any CHAR excluding "[",     ; => may be folded
                "]", "\" & CR, & including
                linear-white-space>
extension-field =
              <Any field which is defined in a document
               published as a formal extension to this
               specification; none will have names beginning
               with the string "X-">
field       =  field-name ":" [ field-body ] CRLF
fields      =    dates                      ; Creation time,
                 source                     ;  author id & one
               1*destination                ;  address required
                *optional-field             ;  others optional
field-body  =  field-body-contents
               [CRLF LWSP-char field-body]
field-body-contents =
```

```
              <the ASCII characters making up the field-body, as
               defined in the following sections, and consisting
               of combinations of atom, quoted-string, and
               specials tokens, or else consisting of texts>
field-name  =  1*<any CHAR, excluding CTLs, SPACE, and ":">
group       =  phrase ":" [#mailbox] ";"
hour        =  2DIGIT ":" 2DIGIT [":" 2DIGIT]
                                          ; 00:00:00 - 23:59:59
HTAB        =  <ASCII HT, horizontal-tab>   ; (     11,       9.)
LF          =  <ASCII LF, linefeed>         ; (     12,      10.)
linear-white-space =  1*([CRLF] LWSP-char)  ; semantics = SPACE
                                          ; CRLF => folding
local-part  =  word *("." word)             ; uninterpreted
                                          ; case-preserved
LWSP-char   =  SPACE / HTAB                 ; semantics = SPACE
mailbox     =  addr-spec                    ; simple address
            /  phrase route-addr            ; name & addr-spec
message     =  fields *( CRLF *text )       ; Everything after
                                          ;  first null line
                                          ;  is message body
month       =  "Jan"  /  "Feb" /  "Mar"  /  "Apr"
            /  "May"  /  "Jun" /  "Jul"  /  "Aug"
            /  "Sep"  /  "Oct" /  "Nov"  /  "Dec"
msg-id      =  "<" addr-spec ">"            ; Unique message id
optional-field =
            /  "Message-ID"        ":"   msg-id
            /  "Resent-Message-ID" ":"   msg-id
            /  "In-Reply-To"       ":"  *(phrase / msg-id)
            /  "References"        ":"  *(phrase / msg-id)
            /  "Keywords"          ":"  #phrase
            /  "Subject"           ":"  *text
            /  "Comments"          ":"  *text
            /  "Encrypted"         ":" 1#2word
            /  extension-field              ; To be defined
            /  user-defined-field           ; May be pre-empted
orig-date   =  "Date"        ":"   date-time
originator  =   authentic                   ; authenticated addr
              [ "Reply-To"   ":" 1#address] )
phrase      =  1*word                       ; Sequence of words
qtext       =  <any CHAR excepting <">,     ; => may be folded
               "\" & CR, and including
               linear-white-space>
quoted-pair =  "\" CHAR                     ; may quote any char
quoted-string = <"> *(qtext/quoted-pair) <">; Regular qtext or
                                          ;   quoted chars.
received    =  "Received"    ":"            ; one per relay
                  ["from" domain]           ; sending host
                  ["by"   domain]           ; receiving host
                  ["via"  atom]             ; physical path
                 *("with" atom)             ; link/mail protocol
                  ["id"   msg-id]           ; receiver msg id
                  ["for"  addr-spec]        ; initial form
                   ";"    date-time         ; time received

resent      =   resent-authentic
              [ "Resent-Reply-To"  ":" 1#address] )
resent-authentic =
```

```
            =   "Resent-From"      ":"   mailbox
            / ( "Resent-Sender"    ":"   mailbox
                "Resent-From"      ":" 1#mailbox  )
resent-date =  "Resent-Date" ":"   date-time
return      =  "Return-path" ":" route-addr ; return address
route       =  1#("@" domain) ":"           ; path-relative
route-addr  =  "<" [route] addr-spec ">"
source      = [  trace ]                    ; net traversals
                 originator                 ; original mail
              [  resent ]                   ; forwarded
SPACE       =  <ASCII SP, space>            ; (     40,      32.)
specials    =  "(" / ")" / "<" / ">" / "@"  ; Must be in quoted-
            /  "," / ";" / ":" / "\" / <">  ;  string, to use
            /  "." / "[" / "]"              ;  within a word.
sub-domain  =  domain-ref / domain-literal
text        =  <any CHAR, including bare    ; => atoms, specials,
                CR & bare LF, but NOT       ;  comments and
                including CRLF>             ;  quoted-strings are
                                            ;  NOT recognized.
time        =  hour zone                    ; ANSI and Military
trace       =    return                     ; path to sender
               1*received                   ; receipt tags
user-defined-field =
             <Any field which has not been defined
              in this specification or published as an
              extension to this specification; names for
              such fields must be unique and may be
              pre-empted by published extensions>
word        =  atom / quoted-string
zone        =  "UT"  / "GMT"                ; Universal Time
                                            ; North American : UT
            /  "EST" / "EDT"                ;  Eastern:  - 5/ - 4
            /  "CST" / "CDT"                ;  Central:  - 6/ - 5
            /  "MST" / "MDT"                ;  Mountain: - 7/ - 6
            /  "PST" / "PDT"                ;  Pacific:  - 8/ - 7
            /  1ALPHA                       ; Military: Z = UT;
<">         =  <ASCII quote mark>           ; (     42,      34.)
```

RFC-826 Address Resolution Protocol
or
Converting Network Protocol Addresses to 48 bit Ethernet Addresses for Transmission on Ethernet Hardware

David C. Plummer
November 1982

This version of RFC-826 incorporates changes and additions made since its original publication.

RFC-826 Address Resolution Protocol

Abstract

The implementation of protocol P on a sending host S decides, through protocol P's routing mechanism, that it wants to transmit to a target host T located some place on a connected piece of 10Mbit Ethernet cable. To actually transmit the Ethernet packet a 48.bit Ethernet address must be generated. The addresses of hosts within protocol P are not always compatible with the corresponding Ethernet address (being different lengths or values). Presented here is a protocol that allows dynamic distribution of the information needed to build tables to translate an address A in protocol P's address space into a 48.bit Ethernet address.

Generalizations have been made which allow the protocol to be used for non-10Mbit Ethernet hardware. Some packet radio networks are examples of such hardware.

The protocol proposed here is the result of a great deal of discussion with several other people, most notably J. Noel Chiappa, Yogen Dalal, and James E. Kulp, and helpful comments from David Moon.

Notes

This protocol was originally designed for the DEC/Intel/Xerox 10Mbit Ethernet. It has been generalized to allow it to be used for other types of networks. Much of the discussion will be directed toward the 10Mbit Ethernet. Generalizations, where applicable, will follow the Ethernet-specific discussion.

DOD Internet Protocol will be referred to as Internet.

Numbers here are in the Ethernet standard, which is high byte first. This is the opposite of the byte addressing of machines such as PDP-11s and VAXes. Therefore, special care must be taken with the opcode field (ar$op) described below.

An agreed upon authority is needed to manage hardware name space values (see below). Until an official authority exists, requests should be submitted to

David C. Plummer
Symbolics, Inc.
243 Vassar Street
Cambridge, Massachusetts 02139
Alternatively, network mail can be sent to DCP@MIT-MC.

RFC-826 Address Resolution Protocol

The Problem

The world is a jungle in general, and the networking game contributes many animals. At nearly every layer of a network architecture there are several potential protocols that could be used. For example, at a high level, there is TELNET and SUPDUP for remote login. Somewhere below that there is a reliable byte stream protocol, which might be CHAOS protocol, DOD TCP, Xerox BSP or DECnet. Even closer to the hardware is the logical transport layer, which might be CHAOS, DOD Internet, Xerox PUP, or DECnet. The 10Mbit Ethernet allows all of these protocols (and more) to coexist on a single cable by means of a type field in the Ethernet packet header. However, the 10Mbit Ethernet requires 48.bit addresses on the physical cable, yet most protocol addresses are not 48.bits long, nor do they necessarily have any relationship to the 48.bit Ethernet address of the hardware. For example, CHAOS addresses are 16.bits, DOD Internet addresses are 32.bits, and Xerox PUP addresses are 8.bits. A protocol is needed to dynamically distribute the correspondences between a <protocol, address> pair and a 48.bit Ethernet address.

Motivation

Use of the 10Mbit Ethernet is increasing as more manufacturers supply interfaces that conform to the specification published by DEC, Intel and Xerox. With this increasing availability, more and more software is being written for these interfaces. There are two alternatives: (1) Every implementor invents his/her own method to do some form of address resolution, or (2) every implementor uses a standard so that his/her code can be distributed to other systems without need for modification. This proposal attempts to set the standard.

Definitions

Define the following for referring to the values put in the TYPE field of the Ethernet packet header:

ether_type$XEROX_PUP (200H),
ether_type$DOD_INTERNET (800H),
ether_type$CHAOS (804H),

and a new one:

ether_type$ADDRESS_RESOLUTION (806H).

Also define the following values (to be discussed later):

ares_op$REQUEST (= 1, high byte transmitted first) and
ares_op$REPLY (= 2),

and

ares_hrd$Ethernet (= 1).

Other hardware types are defined in Assigned Numbers RFC-1060.

RFC-826 Address Resolution Protocol

Packet format

To communicate mappings from <protocol, address> pairs to 48.bit Ethernet addresses, a packet format that embodies the Address Resolution protocol is needed. The format of the packet follows.

Ethernet transmission layer (not necessarily accessible to the user):

48.bit: Ethernet address of destination
48.bit: Ethernet address of sender
16.bit: Protocol type = ether_type$ADDRESS_RESOLUTION

Ethernet packet data:

16.bit: (ar$hrd) Hardware address space (e.g., Ethernet, Packet Radio Net.)
16.bit: (ar$pro) Protocol address space. For Ethernet hardware, this is from the set of type fields ether_typ$<protocol>.
8.bit: (ar$hln) byte length of each hardware address
8.bit: (ar$pln) byte length of each protocol address
16.bit: (ar$op) opcode (ares_op$REQUEST | ares_op$REPLY)
nbytes: (ar$sha) Hardware address of sender of this packet, n from the ar$hln field.
mbytes: (ar$spa) Protocol address of sender of this packet, m from the ar$pln field.
nbytes: (ar$tha) Hardware address of target of this packet (if known).
mbytes: (ar$tpa) Protocol address of target.

Packet Generation

As a packet is sent down through the network layers, routing determines the protocol address of the next hop for the packet and on which piece of hardware it expects to find the station with the immediate target protocol address. In the case of the 10Mbit Ethernet, address resolution is needed and some lower layer (probably the hardware driver) must consult the Address Resolution module (perhaps implemented in the Ethernet support module) to convert the <protocol type, target protocol address> pair to a 48.bit Ethernet address. The Address Resolution module tries to find this pair in a table. If it finds the pair, it gives the corresponding 48.bit Ethernet address back to the caller (hardware driver) which then transmits the packet. If it does not, it probably informs the caller that it is throwing the packet away (on the assumption the packet will be retransmitted by a higher network layer), and generates an Ethernet packet with a type field of ether_type$ADDRESS_RESOLUTION. The Address Resolution module then sets the ar$hrd field to ares_hrd$Ethernet, ar$pro to the protocol type that is being resolved, ar$hln to 6 (the number of bytes in a 48.bit Ethernet address), ar$pln to the length of an address in that protocol, ar$op to ares_op$REQUEST, ar$sha with the 48.bit ethernet address of itself, ar$spa with the protocol address of itself, and ar$tpa with the protocol address of the machine that is trying to be accessed. It does not set ar$tha to anything in particular, because it is this value that it is trying to determine. It could set ar$tha to the broadcast address for the hardware (all ones in the case of the 10Mbit Ethernet) if that makes it convenient for some aspect of the implementation. It then causes this packet to be broadcast to all stations on the Ethernet cable originally determined by the routing mechanism.

Packet Reception

When an address resolution packet is received, the receiving Ethernet module gives the packet to the Address Resolution module which goes through an algorithm similar to the following. Negative conditionals indicate an end of processing and a discarding of the packet.

?Do I have the hardware type in ar$hrd?

Yes: (almost definitely)
[optionally check the hardware length ar$hln]

?Do I speak the protocol in ar$pro?

Yes:
[optionally check the protocol length ar$pln]

Merge_flag := false

If the pair <protocol type, sender protocol address> is already in my translation table, update the sender hardware address field of the entry with the new information in the packet and set Merge_flag to true.

?Am I the target protocol address?

Yes:

If Merge_flag is false, add the triplet <protocol type, sender protocol address, sender hardware address> to the translation table.

?Is the opcode ares_op$REQUEST? (NOW look at the opcode!!)

Yes:

Swap hardware and protocol fields, putting the local hardware and protocol addresses in the sender fields.

Set the ar$op field to ares_op$REPLY

Send the packet to the (new) target hardware address on the same hardware on which the request was received.

Notice that the <protocol type, sender protocol address, sender hardware address> triplet is merged into the table before the opcode is looked at. This is on the assumption that communcation is bidirectional; if A has some reason to talk to B, then B will probably have some reason to talk to A. Notice also that if an entry already exists for the <protocol type, sender protocol address> pair, then the new hardware address supersedes the old one. Related Issues gives some motivation for this.

Generalization: The ar$hrd and ar$hln fields allow this protocol and packet format to be used for non-10Mbit Ethernets. For the 10Mbit Ethernet <arhrd, arhln> takes on the value <1, 6>. For other hardware networks, the ar$pro field may no longer correspond to the Ethernet type field, but it should be associated with the protocol whose address resolution is being sought.

Why is it done this way??

Periodic broadcasting is definitely not desired. Imagine 100 workstations on a single Ethernet, each broadcasting address resolution information once per 10 minutes (as one possible set of parameters). This is one packet every 6 seconds. This is almost reasonable, but what use is it? The workstations aren't generally going to be talking to each other (and therefore have 100 useless entries in a table); they will be mainly talking to a mainframe, file server or bridge, but only to a small number of other workstations (for interactive conversations, for example). The protocol described in this paper distributes information as it is needed, and only once (probably) per boot of a machine.

This format does not allow for more than one resolution to be done in the same packet. This is for simplicity. If things were multiplexed the packet format would be considerably harder to digest, and much of the information could be gratuitous. Think of a bridge that talks four protocols telling a workstation all four protocol addresses, three of which the workstation will probably never use.

This format allows the packet buffer to be reused if a reply is generated; a reply has the same length as a request, and several of the fields are the same.

The value of the hardware field (ar$hrd) is taken from a list for this purpose. Currently the only defined value is for the 10Mbit Ethernet (ares_hrd$Ethernet = 1). There has been talk of using this protocol for Packet Radio Networks as well, and this will require another value as will other future hardware mediums that wish to use this protocol.

For the 10Mbit Ethernet, the value in the protocol field (ar$pro) is taken from the set ether_type$. This is a natural reuse of the assigned protocol types. Combining this with the opcode (ar$op) would effectively halve the number of protocols that can be resolved under this protocol and would make a monitor/debugger more complex (see Network Monitoring and Debugging below). It is hoped that we will never see 32768 protocols, but Murphy made some laws which don't allow us to make this assumption.

In theory, the length fields (ar$hln and ar$pln) are redundant, since the length of a protocol address should be determined by the hardware type (found in ar$hrd) and the protocol type (found in ar$pro). It is included for optional consistency checking, and for network monitoring and debugging (see below).

The opcode is to determine if this is a request (which may cause a reply) or a reply to a previous request. 16 bits for this is overkill, but a flag (field) is needed.

The sender hardware address and sender protocol address are absolutely necessary. It is these fields that get put in a translation table.

The target protocol address is necessary in the request form of the packet so that a machine can determine whether or not to enter the sender information in a table or to send a reply. It is not necessarily needed in the reply form if one assumes a reply is only provoked by a request. It is included for completeness, network monitoring, and to simplify the suggested processing algorithm described above (which does not look at the opcode until AFTER putting the sender information in a table).

The target hardware address is included for completeness and network monitoring. It has no meaning in the request form, since it is this number that the machine is requesting. Its meaning in the reply form is the address of the machine making the request. In some implementations (which do not get to look at the 14.byte ethernet header, for example) this may save some register shuffling or stack space by sending this field to the hardware driver as the hardware destination address of the packet.

There are no padding bytes between addresses. The packet data should be viewed as a byte stream in which only 3 byte pairs are defined to be words (arhrd, arpro and ar$op) which are sent most significant byte first (Ethernet/PDP-10 byte style).

Network Monitoring and Debugging

The above Address Resolution protocol allows a machine to gain knowledge about the higher level protocol activity (e.g., CHAOS, Internet, PUP, DECnet) on an Ethernet cable. It can determine which Ethernet protocol type fields are in use (by value) and the protocol addresses within each protocol type. In fact, it is not necessary for the monitor to speak any of the higher level protocols involved. It goes something like this:

When a monitor receives an Address Resolution packet, it always enters the <protocol type, sender protocol address, sender hardware address> in a table. It can determine the length of the hardware and protocol address from the ar$hln and ar$pln fields of the packet. If the opcode is a REPLY the monitor can then throw the packet away. If the opcode is a REQUEST and the target protocol address matches the protocol address of the monitor, the monitor sends a REPLY as it normally would. The monitor will only get one mapping this way, since the REPLY to the REQUEST will be sent directly to the requesting host. The monitor could try sending its own REQUEST, but this could get two monitors into a REQUEST sending loop, and care must be taken.

Because the protocol and opcode are not combined into one field, the monitor does not need to know which request opcode is associated with which reply opcode for the same higher level protocol. The length fields should also give enough information to enable it to "parse" a protocol addresses, although it has no knowledge of what the protocol addresses mean.

A working implementation of the Address Resolution protocol can also be used to debug a non-working implementation. Presumably a hardware driver will successfully broadcast a packet with Ethernet type field of ether_type$ADDRESS_RESOLUTION. The format of the packet may not be totally correct, because initial implementations may have bugs, and table management may be slightly tricky. Because requests are broadcast a monitor will receive the packet and can display it for debugging if desired.

An Example

Let there exist machines X and Y that are on the same 10Mbit Ethernet cable. They have Ethernet address EA(X) and EA(Y) and DOD Internet addresses IPA(X) and IPA(Y) . Let the Ethernet type of Internet be ET(IP). Machine X has just been started, and sooner or later wants to send an Internet packet to machine Y on the same cable. X knows that it wants to send to IPA(Y) and tells the hardware driver (here an Ethernet driver) IPA(Y). The driver consults the Address Resolution module to convert <ET(IP), IPA(Y)> into a 48.bit Ethernet address, but because X was just started, it does not have this information. It throws the Internet packet away and instead creates an ADDRESS RESOLUTION packet with

(ar$hrd) = ares_hrd$Ethernet
(ar$pro) = ET(IP)
(ar$hln) = length(EA(X))
(ar$pln) = length(IPA(X))
(ar$op) = ares_op$REQUEST
(ar$sha) = EA(X)
(ar$spa) = IPA(X)
(ar$tha) = don't care
(ar$tpa) = IPA(Y)

and broadcasts this packet to everybody on the cable.

Machine Y gets this packet, and determines that it understands the hardware type (Ethernet), that it speaks the indicated protocol (Internet) and that the packet is for it ((ar$tpa)=IPA(Y)). It enters (probably replacing any existing entry) the information that <ET(IP), IPA(X)> maps to EA(X). It then notices that it is a request, so it swaps fields, putting EA(Y) in the new sender Ethernet address field (ar$sha), sets the opcode to reply, and sends the packet directly (not broadcast) to EA(X). At this point Y knows how to send to X, but X still doesn't know how to send to Y.

Machine X gets the reply packet from Y, forms the map from <ET(IP), IPA(Y)> to EA(Y), notices the packet is a reply and throws it away. The next time X's Internet module tries to send a packet to Y on the Ethernet, the translation will succeed, and the packet will (hopefully) arrive. If Y's Internet module then wants to talk to X, this will also succeed since Y has remembered the information from X's request for Address Resolution.

Cache Validation

An implementation of the Address Resolution Protocol (ARP) [LINK:2] **must** provide a mechanism to flush out-of-date cache entries. If this mechanism involves a timeout, it **should** be possible to configure the timeout value.

A mechanism to prevent ARP flooding (repeatedly sending an ARP Request for the same IP address, at a high rate) **must** be included. The recommended maximum rate is 1 per second per destination. (See **Implementation**).

Discussion

The ARP specification [LINK:2] suggests but does not require a timeout mechanism to invalidate cache entries when hosts change their Ethernet addresses. The prevalence of proxy ARP (see Section 2.4 of [INTRO:2]) has significantly increased the likelihood that cache entries in hosts will become invalid, and therefore some ARP-cache invalidation mechanism is now required for hosts. Even in the absence of proxy ARP, a long- period cache timeout is useful in order to automatically correct any bad ARP data that might have been cached.

Cache Validation Implementation

Four mechanisms have been used, sometimes in combination, to flush out-of-date cache entries.

(1) Timeout — Periodically time out cache entries, even if they are in use. Note that this timeout should be restarted when the cache entry is "refreshed" (by observing the source fields, regardless of target address, of an ARP broadcast from the system in question). For proxy ARP situations, the timeout needs to be on the order of a minute.

(2) Unicast Poll — Actively poll the remote host by periodically sending a point-to-point ARP Request to it, and delete the entry if no ARP Reply is received from N successive polls. Again, the timeout should be on the order of a minute, and typically N is 2.

(3) Link-Layer Advice — If the link-layer driver detects a delivery problem, flush the corresponding ARP cache entry.

(4) Higher-layer Advice — Provide a call from the Internet layer to the link layer to indicate a delivery problem. The effect of this call would be to invalidate the corresponding cache entry. This call would be analogous to the "ADVISE_DELIVPROB()" call from the transport layer to the Internet layer (See Internet/Transport Layer Interface), and in fact the ADVISE_DELIVPROB routine might in turn call the link-layer advice routine to invalidate the ARP cache entry.

Approaches (1) and (2) involve ARP cache timeouts on the order of a minute or less. In the absence of proxy ARP, a timeout this short could create noticeable overhead traffic on a very large Ethernet. Therefore, it may be necessary to configure a host to lengthen the ARP cache timeout.

RFC-854 Telnet Protocol Specification

J. Postel & J. Reynolds
USC/Information Sciences Institute
May 1983

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet are expected to adopt and implement this standard. A number of additional options and extensions have been defined since this RFC was originally published; All of these additions have been incorporated into this version.

RFC-854 Telnet Protocol Specification

Introduction

The purpose of the Telnet Protocol is to provide a fairly general, bi-directional, eight-bit byte oriented communications facility. Its primary goal is to allow a standard method of interfacing terminal devices and terminal-oriented processes to each other. It is envisioned that the protocol may also be used for terminal-terminal communication ("linking") and process-process communication (distributed computation).

RFC-854 Telnet Protocol Specification

General Considerations

A Telnet connection is a Transmission Control Protocol (TCP) connection used to transmit data with interspersed Telnet control information.

The Telnet Protocol is built upon three main ideas: first, the concept of a "Network Virtual Terminal"; second, the principle of negotiated options; and third, a symmetric view of terminals and processes.

1. When a Telnet connection is first established, each end is assumed to originate and terminate at a "Network Virtual Terminal", or NVT. An NVT is an imaginary device which provides a standard, network-wide, intermediate representation of a canonical terminal. This eliminates the need for "server" and "user" hosts to keep information about the characteristics of each other's terminals and terminal handling conventions. All hosts, both user and server, map their local device characteristics and conventions so as to appear to be dealing with an NVT over the network, and each can assume a similar mapping by the other party. The NVT is intended to strike a balance between being overly restricted (not providing hosts a rich enough vocabulary for mapping into their local character sets), and being overly inclusive (penalizing users with modest terminals).

   NOTE: The "user" host is the host to which the physical terminal is normally attached, and the "server" host is the host which is normally providing some service. As an alternate point of view, applicable even in terminal-to-terminal or process-to-process communications, the "user" host is the host which initiated the communication.

2. The principle of negotiated options takes cognizance of the fact that many hosts will wish to provide additional services over and above those available within an NVT, and many users will have sophisticated terminals and would like to have elegant, rather than minimal, services. Independent of, but structured within the Telnet Protocol are various "options" that will be sanctioned and may be used with the "DO, DON'T, WILL, WON'T" structure (discussed below) to allow a user and server to agree to use a more elaborate (or perhaps just different) set of conventions for their Telnet connection. Such options could include changing the character set, the echo mode, etc.

   The basic strategy for setting up the use of options is to have either party (or both) initiate a request that some option take effect. The other party may then either accept or reject the request. If the request is accepted the option immediately takes effect; if it is rejected the associated aspect of the connection remains as specified for an NVT. Clearly, a party may always refuse a request to enable, and must never refuse a request to disable some option since all parties must be prepared to support the NVT.

   The syntax of option negotiation has been set up so that if both parties request an option simultaneously, each will see the other's request as the positive acknowledgment of its own.

3. The symmetry of the negotiation syntax can potentially lead to nonterminating acknowledgment loops -- each party seeing the incoming commands not as acknowledgments but as new requests which must be acknowledged. To prevent such loops, the following rules prevail:

   a. Parties may only request a change in option status; i.e., a party may not

send out a "request" merely to announce what mode it is in.

b. If a party receives what appears to be a request to enter some mode it is already in, the request should not be acknowledged. This non-response is essential to prevent endless loops in the negotiation. It is required that a response be sent to requests for a change of mode -- even if the mode is not changed.

c. Whenever one party sends an option command to a second party, whether as a request or an acknowledgment, and use of the option will have any effect on the processing of the data being sent from the first party to the second, then the command must be inserted in the data stream at the point where it is desired that it take effect. (It should be noted that some time will elapse between the transmission of a request and the receipt of an acknowledgment, which may be negative. Thus, a host may wish to buffer data, after requesting an option, until it learns whether the request is accepted or rejected, in order to hide the "uncertainty period" from the user.)

Option Negotiation

Option requests are likely to flurry back and forth when a Telnet connection is first established, as each party attempts to get the best possible service from the other party. Beyond that, however, options can be used to dynamically modify the characteristics of the connection to suit changing local conditions. For example, the NVT, as will be explained later, uses a transmission discipline well suited to the many "line at a time" applications such as BASIC, but poorly suited to the many "character at a time" applications such as NLS. A server might elect to devote the extra processor overhead required for a "character at a time" discipline when it was suitable for the local process and would negotiate an appropriate option. However, rather than then being permanently burdened with the extra processing overhead, it could switch (i.e., negotiate) back to NVT when the detailed control was no longer necessary.

It is possible for requests initiated by processes to stimulate a nonterminating request loop if the process responds to a rejection by merely re-requesting the option. To prevent such loops from occurring, rejected requests should not be repeated until something changes. Operationally, this can mean the process is running a different program, or the user has given another command, or whatever makes sense in the context of the given process and the given option. A good rule of thumb is that a re-request should only occur as a result of subsequent information from the other end of the connection or when demanded by local human intervention.

Option designers should not feel constrained by the somewhat limited syntax available for option negotiation. The intent of the simple syntax is to make it easy to have options -- since it is correspondingly easy to profess ignorance about them. If some particular option requires a richer negotiation structure than possible within "DO, DON'T, WILL, WON'T", the proper tack is to use "DO, DON'T, WILL, WON'T" to establish that both parties understand the option, and once this is accomplished a more exotic syntax can be used freely. For example, a party might send a request to alter (establish) line length. If it is accepted, then a different syntax can be used for actually negotiating the line length -- such a "sub-negotiation" might include fields for minimum allowable, maximum allowable and desired line lengths. The important concept is that such expanded negotiations should never begin until some prior (standard) negotiation has established that both parties are capable of parsing the expanded syntax.

In summary, WILL XXX is sent, by either party, to indicate that party's desire (offer) to begin performing option XXX, DO XXX and DON'T XXX being its positive and negative acknowledgments; similarly, DO XXX is sent to indicate a desire (request) that the other party (i.e., the recipient of the DO) begin performing option XXX, WILL XXX and WON'T XXX being the positive and negative acknowledgments. Since the NVT is what is left when no options are enabled, the DON'T and WON'T responses are guaranteed to leave the connection in a state which both ends can handle. Thus, all hosts may implement their Telnet processes to be totally unaware of options that are not supported, simply returning a rejection to (i.e., refusing) any option request that cannot be understood.

For options that require a more information than a single option code a subnegotiation mechanism is provided.

As much as possible, the Telnet protocol has been made server-user symmetrical so that it easily and naturally covers the user-user (linking) and server-server (cooperating processes) cases. It is hoped, but not absolutely required, that options will further this intent. In any case, it is explicitly acknowledged that symmetry is an operating principle rather than an ironclad rule.

A companion document, "Telnet Option Specification," (RFC-855) should be consulted for

information about the procedure for establishing new options.

RFC-854 Telnet Protocol Specification

The Network Virtual Terminal

The Network Virtual Terminal (NVT) is a bi-directional character device. The NVT has a printer and a keyboard. The printer responds to incoming data and the keyboard produces outgoing data which is sent over the Telnet connection and, if "echoes" are desired, to the NVT's printer as well. "Echoes" will not be expected to traverse the network (although options exist to enable a "remote" echoing mode of operation, no host is required to implement this option). The code set is seven-bit USASCII in an eight-bit field, except as modified herein. Any code conversion and timing considerations are local problems and do not affect the NVT.

Transmission Of Data

Although a Telnet connection through the network is intrinsically full duplex, the NVT is to be viewed as a half-duplex device operating in a line-buffered mode. That is, unless and until options are negotiated to the contrary, the following default conditions pertain to the transmission of data over the Telnet connection:

1) Insofar as the availability of local buffer space permits, data should be accumulated in the host where it is generated until a complete line of data is ready for transmission, or until some locally-defined explicit signal to transmit occurs. This signal could be generated either by a process or by a human user.

   The motivation for this rule is the high cost, to some hosts, of processing network input interrupts, coupled with the default NVT specification that "echoes" do not traverse the network. Thus, it is reasonable to buffer some amount of data at its source. Many systems take some processing action at the end of each input line (even line printers or card punches frequently tend to work this way), so the transmission should be triggered at the end of a line. On the other hand, a user or process may sometimes find it necessary or desirable to provide data which does not terminate at the end of a line; therefore implementers are cautioned to provide methods of locally signaling that all buffered data should be transmitted immediately.

2) When a process has completed sending data to an NVT printer and has no queued input from the NVT keyboard for further processing (i.e., when a process at one end of a Telnet connection cannot proceed without input from the other end), the process must transmit the **Telnet Go Ahead** (GA) command.

   This rule is not intended to require that the Telnet GA command be sent from a terminal at the end of each line, since server hosts do not normally require a special signal (in addition to end-of-line or other locally-defined characters) in order to commence processing. Rather, the Telnet GA is designed to help a user's local host operate a physically half duplex terminal which has a "lockable" keyboard such as the IBM 2741. A description of this type of terminal may help to explain the proper use of the GA command.

   The terminal-computer connection is always under control of either the user or the computer. Neither can unilaterally seize control from the other; rather the controlling end must relinguish its control explicitly. At the terminal end, the hardware is constructed so as to relinquish control each time that a "line" is terminated (i.e., when the "New Line" key is typed by the user). When this occurs, the attached (local) computer processes the input data, decides if output should be generated, and if not returns control to the terminal. If output should be generated, control is retained by the computer until all output has been transmitted.

   The difficulties of using this type of terminal through the network should be obvious. The "local" computer is no longer able to decide whether to retain control after seeing an end-of-line signal or not; this decision can only be made by the "remote" computer which is processing the data. Therefore, the Telnet GA command provides a mechanism whereby the "remote" (server) computer can signal the "local" (user) computer that it is time to pass control to the user of the terminal. It should be transmitted at those times, and only at those times, when the user should be given control of the terminal. Note

that premature transmission of the GA command may result in the blocking of output, since the user is likely to assume that the transmitting system has paused, and therefore he will fail to turn the line around manually.

The foregoing, of course, does not apply to the user-to-server direction of communication. In this direction, GAs may be sent at any time, but need not ever be sent. Also, if the Telnet connection is being used for process-to-process communication, GAs need not be sent in either direction. Finally, for terminal-to-terminal communication, GAs may be required in neither, one, or both directions. If a host plans to support terminal-to-terminal communication it is suggested that the host provide the user with a means of manually signaling that it is time for a GA to be sent over the Telnet connection; this, however, is not a requirement on the implementer of a Telnet process.

Note that the symmetry of the Telnet model requires that there is an NVT at each end of the Telnet connection, at least conceptually.

RFC-854 Telnet Protocol Specification - Transmission of Data

Telnet Go-Ahead Function

On a host that never sends the Telnet command Go Ahead (GA), the Telnet Server **must** attempt to negotiate the Suppress Go Ahead option (i.e., send "WILL Suppress Go Ahead"). A User or Server Telnet **must** always accept negotiation of the Suppress Go Ahead option.

When it is driving a full-duplex terminal for which GA has no meaning, a User Telnet implementation MAY ignore GA commands.

> **Discussion**
> Half-duplex ("locked-keyboard") line-at-a-time terminals for which the Go-Ahead mechanism was designed have largely disappeared from the scene. Itturned out to be difficult to implement sending the Go-Ahead signal in many operating systems, even some systems that support native half-duplex terminals. The difficulty is typically that the Telnet server code does not have access to information about whether the user process is blocked awaiting input from the Telnet connection, i.e., it cannot reliably determine when to send a GA command. Therefore, most Telnet Server hosts do not send GA commands.
>
> The effect of the rules in this section is to allow either end of a Telnet connection to veto the use of GA commands.
>
> There is a class of half-duplex terminals that is still commercially important: "data entry terminals," which interact in a full-screen manner. However, supporting data entry terminals using the Telnet protocol does not require the Go Ahead signal; see Data Entry Terminals.

Standard Representation Of Control Functions

As stated in the Introduction to this document, the primary goal of the Telnet protocol is the provision of a standard interfacing of terminal devices and terminal-oriented processes through the network. Early experiences with this type of interconnection have shown that certain functions are implemented by most servers, but that the methods of invoking these functions differ widely. For a human user who interacts with several server systems, these differences are highly frustrating. Telnet, therefore, defines a standard representation for five of these functions, as described below. These standard representations have standard, but not required, meanings (with the exception that the Interrupt Process (IP) function may be required by other protocols which use Telnet); that is, a system which does not provide the function to local users need not provide it to network users and may treat the standard representation for the function as a No-operation. On the other hand, a system which does provide the function to a local user is obliged to provide the same function to a network user who transmits the standard representation for the function.

Interrupt Process (IP)
Many systems provide a function which suspends, interrupts, aborts, or terminates the operation of a user process. This function is frequently used when a user believes his process is in an unending loop, or when an unwanted process has been inadvertently activated. IP is the standard representation for invoking this function. A server Telnet is required to support the Telnet IP function even if the server host has an equivalent in-stream function (e.g., Control-C in many systems). The Telnet IP function may be stronger than an in-stream interrupt command, because of the out-of-band effect of TCP urgent data.

It should be noted by implementers that IP may be required by other protocols which use Telnet, and therefore should be implemented if these other protocols are to be supported.

A User Telnet **should** have the capability of flushing output when it sends a Telnet IP.

Abort Output (AO)
Many systems provide a function which allows a process, which is generating output, to run to completion (or to reach the same stopping point it would reach if running to completion) but without sending the output to the user's terminal. Further, this function typically clears any output already produced but not yet actually printed (or displayed) on the user's terminal. AO is the standard representation for invoking this function. For example, some subsystem might normally accept a user's command, send a long text string to the user's terminal in response, and finally signal readiness to accept the next command by sending a "prompt" character (preceded by <CR><LF>) to the user's terminal. If the AO were received during the transmission of the text string, a reasonable implementation would be to suppress the remainder of the text string, but transmit the prompt character and the preceding <CR><LF>. (This is possibly in distinction to the action which might be taken if an IP were received; the IP might cause suppression of the text string and an exit from the subsystem.)

It should be noted, by server systems which provide this function, that there may be buffers external to the system (in the network and the user's local host) which should be cleared; the appropriate way to do this is to transmit the "Synch" signal (described below) to the user system.

Are You There (AYT)
Many systems provide a function which provides the user with some visible (e.g., printable) evidence that the system is still up and running. This function may be invoked by the user when the system is unexpectedly "silent" for a long time, because of the unanticipated (by the user) length of a computation, an unusually heavy system load, etc. AYT is the standard

representation for invoking this function.

Erase Character (EC)
Many systems provide a function which deletes the last preceding undeleted character or "print position"* from the stream of data being supplied by the user. This function is typically used to edit keyboard input when typing mistakes are made. EC is the standard representation for invoking this function.

> *NOTE: A "print position" may contain several characters which are the result of overstrikes, or of sequences such as <char1> BS <char2>...

Erase Line (EL)
Many systems provide a function which deletes all the data in the current "line" of input. This function is typically used to edit keyboard input. EL is the standard representation for invoking this function.

End-of-Record(EOR)
The EOR control function may be used to delimit the stream. An important application is data entry terminal support.

RFC-854 Telnet Protocol Specification

The Telnet "Synch" Signal

Most time-sharing systems provide mechanisms which allow a terminal user to regain control of a "runaway" process; the IP and AO functions described above are examples of these mechanisms. Such systems, when used locally, have access to all of the signals supplied by the user, whether these are normal characters or special "out of band" signals such as those supplied by the teletype "BREAK" key or the IBM 2741 "ATTN" key. This is not necessarily true when terminals are connected to the system through the network; the network's flow control mechanisms may cause such a signal to be buffered elsewhere, for example in the user's host.

To counter this problem, the Telnet "Synch" mechanism is introduced. A Synch signal consists of a TCP Urgent notification, coupled with the Telnet command DATA MARK. The Urgent notification, which is not subject to the flow control pertaining to the Telnet connection, is used to invoke special handling of the data stream by the process which receives it. In this mode, the data stream is immediately scanned for "interesting" signals as defined below, discarding intervening data. The Telnet command DATA MARK (DM) is the synchronizing mark in the data stream which indicates that any special signal has already occurred and the recipient can return to normal processing of the data stream.

> The Synch is sent via the TCP send operation with the Urgent flag set and the DM as the last (or only) data octet.

When several Synchs are sent in rapid succession, the Urgent notifications may be merged. It is not possible to count Urgents since the number received will be less than or equal the number sent. When in normal mode, a DM is a no operation; when in urgent mode, it signals the end of the urgent processing.

> If TCP indicates the end of Urgent data before the DM is found, Telnet should continue the special handling of the data stream until the DM is found.
>
> If TCP indicates more Urgent data after the DM is found, it can only be because of a subsequent Synch. Telnet should continue the special handling of the data stream until another DM is found.

"Interesting" signals are defined to be: the Telnet standard representations of IP, AO, and AYT (but not EC or EL); the local analogs of these standard representations (if any); all other Telnet commands; other site-defined signals which can be acted on without delaying the scan of the data stream.

Since one effect of the SYNCH mechanism is the discarding of essentially all characters (except Telnet commands) between the sender of the Synch and its recipient, this mechanism is specified as the standard way to clear the data path when that is desired. For example, if a user at a terminal causes an AO to be transmitted, the server which receives the AO (if it provides that function at all) should return a Synch to the user.

Finally, just as the TCP Urgent notification is needed at the Telnet level as an out-of-band signal, so other protocols which make use of Telnet may require a Telnet command which can be viewed as an out-of-band signal at a different level.

By convention the sequence [IP, Synch] is to be used as such a signal. For example, suppose that some other protocol, which uses Telnet, defines the character string STOP analogously to the Telnet command AO. Imagine that a user of this protocol wishes a server to process the STOP string, but the connection is blocked because the server is processing other commands. The user should instruct his system to:

1. Send the Telnet IP character;

2. Send the Telnet SYNC sequence, that is:

   Send the Data Mark (DM) as the only character in a TCP urgent mode send operation.

3. Send the character string STOP; and
4. Send the other protocol's analog of the Telnet DM, if any.

The user (or process acting on his behalf) must transmit the Telnet SYNCH sequence of step 2 above to ensure that the Telnet IP gets through to the server's Telnet interpreter.

> The Urgent should wake up the Telnet process; the IP should wake up the next higher level process.

When it receives a Telnet AO command, a Server Telnet **must** send a Telnet "Synch" sequence back to the user, to flush the output stream.

Flushing Output

A User Telnet program **should** provide the user the ability to specify whether or not output should be flushed when an IP is sent.

For any output flushing scheme that causes the User Telnet to flush output locally until a Telnet signal is received from the Server, there **should** be a way for the user to manually restore normal output, in case the Server fails to send the expected signal.

There are three possible ways for a User Telnet to flush the stream of server output data:

(1) Send AO after IP.

This will cause the server host to send a "flush-buffered-output" signal to its operating system. However, the AO may not take effect locally, i.e., stop terminal output at the User Telnet end, until the Server Telnet has received and processed the AO and has sent back a "Synch".

(2) Send DO TIMING-MARK [TELNET:7] after IP, and discard all output locally until a WILL/WONT TIMING-MARK is received from the Server Telnet.

Since the DO TIMING-MARK will be processed after the IP at the server, the reply to it should be in the right place in the output data stream. However, the TIMING-MARK will not send a "flush buffered output" signal to the server operating system. Whether or not this is needed is dependent upon the server system.

(3) Do both.

The best method is not entirely clear, since it must accommodate a number of existing server hosts that do not follow the Telnet standards in various ways. The safest approach is probably to provide a user-controllable option to select (1), (2), or (3).

The NVT Printer and Keyboard

The NVT printer has an unspecified carriage width and page length and can produce representations of all 95 USASCII graphics (codes 32 through 126). Of the 33 USASCII control codes (0 through 31 and 127), and the 128 uncovered codes (128 through 255), the following have specified meaning to the NVT printer:

Name	Code	Meaning
NULL (NUL)	0	No Operation
Line Feed (LF)	10	Moves the printer to the next print line, keeping the same horizontal position.
Carriage Return (CR)	13	Moves the printer to the left margin of the current line.

In addition, the following codes shall have defined, but not required, effects on the NVT printer. Neither end of a Telnet connection may assume that the other party will take, or will have taken, any particular action upon receipt or transmission of these:

BELL (BEL)	7	Produces an audible or visible signal (which does NOT move the print head).
Back Space (BS)	8	Moves the print head one character position towards the left margin.
Horizontal Tab (HT)	9	Moves the printer to the next horizontal tab stop. It remains unspecified how either party determines or establishes where such tab stops are located.
Vertical Tab (VT)	11	Moves the printer to the next vertical tab stop. It remains unspecified how either party determines or establishes where such tab stops are located.
Form Feed (FF)	12	Moves the printer to the top of the next page, keeping the same horizontal position.

All remaining codes do not cause the NVT printer to take any action.

The sequence "CR LF", as defined, will cause the NVT to be positioned at the left margin of the next print line (as would, for example, the sequence "LF CR"). However, many systems and terminals do not treat CR and LF independently, and will have to go to some effort to simulate their effect. (For example, some terminals do not have a CR independent of the LF, but on such terminals it may be possible to simulate a CR by backspacing.) Therefore, the sequence "CR LF" must be treated as a single "new line" character and used whenever their combined action is intended; the sequence "CR NUL" must be used where a carriage return alone is actually desired; and the CR character must be avoided in other contexts. This rule gives assurance to systems which must decide whether to perform a "new line" function or a multiple-backspace that the Telnet stream contains a character following a CR that will allow a rational decision.

> Note that "CR LF" or "CR NUL" is required in both directions (in the default ASCII mode), to preserve the symmetry of the NVT model. Even though it may be known in some situations (e.g., with remote echo and suppress go ahead options in effect) that characters are not being sent to an actual printer, nonetheless, for the sake of consistency, the protocol requires that a NUL be inserted following a CR not followed by a LF in the data stream. The converse of this is that a NUL received in the data stream after a CR (in the absence of options negotiations which explicitly specify otherwise) should be stripped out prior to applying the NVT to local character set mapping.

The NVT keyboard has keys, or key combinations, or key sequences, for generating all 128 USASCII codes. Note that although many have no effect on the NVT printer, the NVT keyboard is capable of generating them.

In addition to these codes, the NVT keyboard shall be capable of generating the following additional codes which, except as noted, have defined, but not reguired, meanings. The actual code assignments for these "characters" are in the Telnet Command section, because they are viewed as being, in some sense, generic and should be available even when the data stream is interpreted as being some other character set.

Synch
This key allows the user to clear his data path to the other party. The activation of this key causes a DM (see command section) to be sent in the data stream and a TCP Urgent notification is associated with it. The pair DM-Urgent is to have required meaning as defined previously.

Break (BRK)
This code is provided because it is a signal outside the USASCII set which is currently given local meaning within many systems. It is intended to indicate that the Break Key or the Attention Key was hit. Note, however, that this is intended to provide a 129th code for systems which require it, not as a synonym for the IP standard representation.

Interrupt Process (IP)
Suspend, interrupt, abort or terminate the process to which the NVT is connected. Also, part of the out-of-band signal for other protocols which use Telnet.

Abort Output (AO)
Allow the current process to (appear to) run to completion, but do not send its output to the user. Also, send a Synch to the user.

Are You There (AYT)
Send back to the NVT some visible (i.e., printable) evidence that the AYT was received.

Erase Character (EC)
The recipient should delete the last preceding undeleted character or "print position" from the data stream.

Erase Line (EL)
The recipient should delete characters from the data stream back to, but not including, the last "CR LF" sequence sent over the Telnet connection.

The spirit of these "extra" keys, and also the printer format effectors, is that they should represent a natural extension of the mapping that already must be done from "NVT" into "local". Just as the NVT data byte 68 (104 octal) should be mapped into whatever the local code for "uppercase D" is, so the EC character should be mapped into whatever the local "Erase Character" function is. Further, just as the mapping for 124 (174 octal) is somewhat arbitrary in an environment that has no "vertical bar" character, the EL character may have a somewhat arbitrary mapping (or none at all) if there is no local "Erase Line" facility. Similarly for format effectors: if the terminal actually does have a "Vertical Tab", then the mapping for VT is obvious, and only when the terminal does not have a vertical tab should the effect of VT be unpredictable.

In NVT mode, a Telnet **should not** send characters with the high-order bit 1, and **must not** send it as a parity bit. Implementations that pass the high-order bit to applications **should** negotiate binary mode

Discussion
Implementors should be aware that a strict reading of the above specification

allows a client or server expecting NVT ASCII to ignore characters with the high-order bit set. In general, binary mode is expected to be used for transmission of an extended (beyond 7-bit) character set with Telnet.

However, there exist applications that really need an 8- bit NVT mode, which is currently not defined, and these existing applications do set the high-order bit during part or all of the life of a Telnet connection. Note that binary mode is not the same as 8-bit NVT mode, since binary mode turns off end-of-line processing. For this reason, the requirements on the high-order bit are stated as **should**, not **must**.

The specification defines a minimal set of properties of a "network virtual terminal" or NVT; this is not meant to preclude additional features in a real terminal. A Telnet connection is fully transparent to all 7-bit ASCII characters, including arbitrary ASCII control characters.

For example, a terminal might support full-screen commands coded as ASCII escape sequences; a Telnet implementation would pass these sequences as uninterpreteddata. Thus, an NVT should not be conceived as a terminal type of a highly-restricted device..

RFC-854 Telnet Protocol Specification

Telnet End-of-Line Convention

The Telnet protocol defines the sequence CR LF to mean "end-of-line". For terminal input, this corresponds to a command- completion or "end-of-line" key being pressed on a user terminal; on an ASCII terminal, this is the CR key, but it may also be labelled "Return" or "Enter".

When a Server Telnet receives the Telnet end-of-line sequence CR LF as input from a remote terminal, the effect **must** be the same as if the user had pressed the"end-of-line" key on a local terminal. On server hosts that use ASCII, in particular, receipt of the Telnet sequence CR LF must cause the same effect as a local user pressing the CR key on a local terminal. Thus, CR LF and CR NUL **must** have the same effect on an ASCII server host when received as input over a Telnetconnection.

A User Telnet **must** be able to send any of the forms: CR LF, CR NUL, and LF. A User Telnet on an ASCII host **should** have a user-controllable mode to send eitherCR LF or CR NUL when the user presses the "end-of-line" key, and CR LF **should** be the default.

The Telnet end-of-line sequence CR LF **must** be used to send Telnet data that is not terminal-to-computer (e.g., for Server Telnet sending output, or the Telnet protocol incorporated another application protocol).

RFC-854 Telnet Protocol Specification - End-of-Line Convention

Discussion

To allow interoperability between arbitrary Telnet clients and servers, the Telnet protocol defined a standard representation for a line terminator. Since the ASCII character set includes no explicit end-of-line character, systems have chosen various representations, e.g., CR, LF, and the sequence CR LF.The Telnet protocol chose the CR LF sequence as the standard for network transmission.

Unfortunately, the Telnet protocol specification has turned out to be somewhat ambiguous on what character(s) should be sent from clientto server for the "end-of-line" key. The result has been a massive and continuing interoperability headache, made worse by various faulty implementations of both User and Server Telnets.

Although the Telnet protocol is based on a perfectly symmetric model, in a remote login session the role of the user at a terminal differs from the role of the server host. For example, the original specification defines the meaning of CR, LF, and CR LF as output from the server, but does not specify what the User Telnet should send when the user presses the "end-of-line" key on the terminal; this turns out to be the point at issue.

When a user presses the "end-of-line" key, some User Telnet implementations send CR LF, while others send CR NUL. These will be equivalent for a correctly-implemented ASCII server host, as discussed above. For other servers, a mode in the User Telnet is needed.

The existence of User Telnets that send only CR NUL when CR is pressed creates a dilemma for non-ASCII hosts: they can either treat CR NUL as equivalent to CR LF in input, thus precluding the possibility of entering a "bare" CR, or else lose complete interworking.

Suppose a user on host A uses Telnet to log into a server host B, and then execute B's User Telnet program to log into server host C. It is desirable for the Server/User Telnet combination on B to be as transparent as possible, i.e., to appear as if A were connected directly to C. In particular, correct implementation will make B transparent to Telnet end-of-line sequences, except that CR LF may be translated to CR NUL or vice versa.

RFC-854 Telnet Protocol Specification - End-of-Line Convention

Implementation

To understand Telnet end-of-line issues, one must have at least a general model of the relationship of Telnet to the local operating system. The Server Telnet process is typically coupled into the terminal driver software of the operating system as a pseudo-terminal. A Telnet end-of- line sequence received by the Server Telnet must have the same effect as pressing the end-of-line key on a real locally-connected terminal.

Operating systems that support interactive character-at- a-time applications (e.g., editors) typically have two internal modes for their terminal I/O: a formatted mode, in which local conventions for end-of-line and other formatting rules have been applied to the data stream, and a "raw" mode, in which the application has direct access to every character as it was entered. A Server Telnet must be implemented in such a way that these modes have the same effect for remote as for local terminals. For example, suppose a CR LF or CR NUL is received by the Server Telnet on an ASCII host. In raw mode, a CR character is passed to theapplication; in formatted mode, the local system's end-of-line convention is used.

Data Entry Terminals

In addition to the line-oriented and character-oriented ASCII terminals for which Telnet was designed, there are several families of video display terminals that are sometimes known as "data entry terminals" or DETs. The IBM 3270 family is a well-known example.

Two Internet protocols have been designed to support generic DETs: SUPDUP Protocol and SUPDUP Option [RFC-734, RFC-736], and the DET option [RFC-732]. The DET option drives a data entry terminal over a Telnet connection using (sub-) negotiation. SUPDUP is a completely separate terminal protocol, which can be entered from Telnet by negotiation. Although both SUPDUP and the DET option have been used successfully in particular environments, neither has gained general acceptance or wide implementation.

A different approach to DET interaction has been developed for supporting the IBM 3270 family through Telnet, although the same approach would be applicable toany DET. The idea is to enter a "native DET" mode, in which the native DET input/output stream is sent as binary data. The Telnet EOR command is used to delimit logical records (e.g., "screens") within this binary stream.

Implementation
The rules for entering and leaving native DET mode are as follows:

- The Server uses the Terminal-Type option (RFC-1091) to learn that the client is a DET.
- It is conventional, but not required, that both ends negotiate the EOR option [RFC-885].
- Both ends negotiate the Binary option [RFC-856] to enter native DET mode.
- When either end negotiates out of binary mode, the other end does too, and the mode then reverts to normal NVT.

RFC-854 Telnet Protocol Specification

Telnet Command Structure

All Telnet commands consist of at least a two byte sequence: the "Interpret as Command" (IAC) escape character followed by the code for the command. The commands dealing with option negotiation are three byte sequences, the third byte being the code for the option referenced. This format was chosen so that as more comprehensive use of the "data space" is made -- by negotiations from the basic NVT, of course -- collisions of data bytes with reserved command values will be minimized, all such collisions requiring the inconvenience, and inefficiency, of "escaping" the data bytes into the stream. With the current set-up, only the IAC need be doubled to be sent as data, and the other 255 codes may be passed transparently.

The following are the defined Telnet commands. Note that these codes and code sequences have the indicated meaning only when immediately preceded by an IAC.

Note: Each Option was originally specified in a separate RFC. The format of these RFCs followed the guidelines of RFC-855 "Telnet Option Specification". Some of the text of the original specifications has been dropped from the references that follow.

Name	Code	Meaning
Binary	0	Transmit Binary
Echo	1	Perform Local Echoing
Reconnect	2	Negotiate Process Reconnection
Suppress GA	3	Suppress sending Go-Ahead commands
Message Size	4	Approximate Message Size
Status	5	Request Status of Connection
TM	6	Timing-Mark
Remote	7	Remote Controlled Transmit and Echo
Line Width	8	
Page Size	9	
CR Disp	10	Carriage-Return Disposition
HT Stops	11	Horizontal Tab Stops
HT Disp	12	Horizontal Tab Disposition
FF Disp	13	Form Feed Disposition
VT Stops	14	Verical Tab Stops
VT Disp	15	Vertical Tab Disposition
LF Disp	16	Linefeed Disposition
Ext ASCII	17	Extended ASCII character set
Logout	18	End Remote Session
Byte Macro	19	Byte Macro Option
DET	20	Data Entry Terminal
SUPDUP	21	SUPDUP Protocol
SUPDUP-Output	22	SUPDUP-Output Option
Send Loc	23	Send Location
T-Type	24	Terminal Type
EOR	25	End-of-Record
TACACS	26	TACACS User Indentification
Output Mark	27	Output Marking
Terminal Loc	28	Terminal Location Number
X.3 Pad	30	X.3 Pad Option
Negotiate Window	31	Negotiate Window Size Option
Terminal Speed	32	Negotiate Terminal Speed
Remote Flow Ctl	33	Negotiate Remote Flow Control

Linemode	34	Linemode Option
X Display Location	35	X Display Location Option
SE	240	End of subnegotiation parameters.
NOP	241	No operation.
Data Mark	242	The data stream portion of a Synch. This should always be accompanied by a TCP Urgent notification.
Break	243	NVT character BRK.
Interrupt Process	244	The function IP.
Abort output	245	The function AO.
Are You There	246	The function AYT.
Erase character	247	The function EC.
Erase Line	248	The function EL.
Go ahead	249	The GA signal.
SB	250	Indicates that what follows is subnegotiation of the indicated option.
WILL (option code)	251	Indicates the desire to begin performing, or confirmation that you are now performing, the indicated option.
WON'T (option code)	252	Indicates the refusal to perform, or continue performing, the indicated option.
DO (option code)	253	Indicates the request that the other party perform, or confirmation that you are expecting the other party to perform, the indicated option.
DON'T (option code)	254	Indicates the demand that the other party stop performing, or confirmation that you are no longer expecting the other party to perform, the indicated option.
IAC or EXOPL	255	Data Byte 255.

Option Requirements

Every Telnet implementation **must** support the Binary option [RFC-856] and the Suppress Go Ahead option [RFC-858], and **should** support the Echo [RFC-857], Status [RFC-859], End-of-Record [RFC-885], and Extended Options List [RFC-861] options.

A User or Server Telnet **should** support the Window Size Option [RFC-1073] if the local operating system provides the corresponding capability.

Discussion
Note that the End-of-Record option only signifies that a Telnet can receive a Telnet EOR without crashing; therefore, every Telnet ought to be willing to accept negotiation of the End-of-Record option.

Option Initiation

When the Telnet protocol is used in a client/server situation, the server **should** initiate negotiation of the terminal interaction mode it expects.

> **Discussion**
> The Telnet protocol was defined to be perfectly symmetrical, but its application is generally asymmetric. Remote login has been known to fail because **neither** side initiated negotiation of the required non-default terminal modes. It is generally the server that determines the preferred mode, so the server needs to initiate the negotiation; since the negotiation is symmetric, the user can also initiate it.

A client (User Telnet) **should** provide a means for users to enable and disable the initiation of option negotiation.

> **Discussion**
> A user sometimes needs to connect to an application service (e.g., FTP or SMTP) that uses Telnet for its control stream but does not support Telnet options. User Telnet may be used for this purpose if initiation of option negotiation is disabled.

RFC-854 Telnet Protocol Specification

Linemode Option

An important new Telnet option, Linemode [RFC-1116], has been proposed. The **Linemode** option provides a standard way for a User Telnet and a Server Telnet to agree that the client rather than the server will perform terminal character processing. When the client has prepared a complete line of text, it will send it to the server in (usually) one TCP packet. This option will greatly decrease the packet cost of Telnet sessions and will also give much better user response over congested or long- delay networks.

> The **Linemode** option allows dynamic switching between local and remote character processing. For example, the Telnet connection will automatically negotiate into single- character mode while a full screen editor is running, and then return to linemode when the editor is finished.
>
> We expect that when this RFC is released, hosts should implement the client side of this option, and may implement the server side of this option. To properlyimplement the server side, the server needs to be able to tell the local system not to do any input character processing, but to remember its current terminal state and notify the Server Telnet process whenever the state changes. This will allow password echoing and full screen editors to be handled properly, for example.

Connection Establishment

The Telnet TCP connection is established between the user's port U and the server's port L. The server listens on its well known port L for such connections. Since a TCP connection is full duplex and identified by the pair of ports, the server can engage in many simultaneous connections involving its port L and different user ports U.

Port Assignment

When used for remote user access to service hosts (i.e., remote terminal access) this protocol is assigned server port 23 (27 octal). That is L=23.

Character Set Transparency

User Telnet implementations **should** be able to send or receive any 7-bit ASCII character. Where possible, any special character interpretations by the user host's operating system **should** be bypassed so that these characters can conveniently be sent and received on the connection.

Some character value **must** be reserved as "escape to command mode"; conventionally, doubling this character allows it to be entered as data. The specific character used **should** be user selectable.

On binary-mode connections, a User Telnet program MAY provide an escape mechanism for entering arbitrary 8-bit values, if the host operating system doesn't allow them to be entered directly from the keyboard.

> **Implementation**
> The transparency issues are less pressing on servers, but implementors should take care in dealing with issues like: masking off parity bits (sent by an older, non-conforming client) before they reach programs that expect only NVT ASCII, and properly handling programs that request 8-bit data streams.

User Telnet Capabilities

A User Telnet program **should** allow the user to optionally specify a non-standard contact port number at the Server Telnet host.

A User Telnet program **must** provide a user the capability of entering any of the Telnet control functions IP, AO, or AYT, and **should** provide the capability of entering EC, EL, and Break.

A User Telnet program **should** report to the user any TCP errors that are reported by the transport layer (see TCP/Application Layer Interface).

Abort Output -- Allow the current process to (appear to) run to completion, but do not send its output to the user. Also, send a SYNCH to the user.

Are You There -- Send back to the NVT some visible (i.e. printable) evidence that the **AYT** was received.

Break -- This code is provided because it is a signal outside the USASCII set which is currently given local meaning within many systems. It is intended to indicate that the Break Key or the Attention Key was hit. Note, however, that this is intended to provide a 129th code for systems which require it, not as a synonym forthe IP standard representation.

Data Mark -- The data stream portion of a Synch. This should always be accompanied by a TCP Urgent notification.

Erase Character -- Invoke the system specific function to delete the last preceding undeleted character or "print position".

Erase Line -- Invoke the system specific funtion to delete all the data in the current "line" of input.

Interrupt Process -- Suspend, interrupt, abort or terminate the process to which the NVT is connected. Also, part of the out-of-band signal for other protocols which use TELNET.

Subnegotiation Begin -- Indicates that what follows is subnegotiation of the indicated option.

Subnegotiation End -- Indicates the end of subnegotiation parameters.

RFC-855 Telnet Option Specifications

J. Postel & J. Reynolds
USC/Information Sciences Institute
May 1983

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet are expected to adopt and implement this standard.

The intent of providing for options in the TELNET Protocol is to permit hosts to obtain more elegant solutions to the problems of communication between dissimilar devices than is possible within the framework provided by the Network Virtual Terminal (NVT). It should be possible for hosts to invent, test, or discard options at will. Nevertheless, it is envisioned that options which prove to be generally useful will eventually be supported by many hosts; therefore it is desirable that options should be carefully documented and well publicized. In addition, it is necessary to insure that a single option code is not used for several different options.

This document specifies a method of option code assignment and standards for documentation of options. The individual responsible for assignment of option codes may waive the requirement for complete documentation for some cases of experimentation, but in general documentation will be required prior to code assignment. Options will be publicized by publishing their documentation as RFCs; inventors of options may, of course, publicize them in other ways as well.

Option codes will be assigned by:

Jonathan B. Postel
University of Southern California
Information Sciences Institute (USC-ISI)
4676 Admiralty Way
Marina Del Rey, California 90291
(213) 822-1511

Mailbox = POSTEL@ISI.EDU

Documentation of options should contain at least the following sections:

Section 1 - Command Name and Option Code

Section 2 - Command Meanings

The meaning of each possible TELNET command relevant to this option should be described. Note that for complex options, where "subnegotiation" is required, there may be a larger number of possible commands.

Section 3 - Default Specification

The default assumptions for hosts which do not implement, or use, the option must be described.

Section 4 - Motivation

A detailed explanation of the motivation for inventing a particular option, or for choosing a particular form for the option, is extremely helpful to those who are not faced (or don't realize that they are faced) by the problem that the option is designed to solve.

Section 5 - Description (or Implementation Rules)

Merely defining the command meanings and providing a statement of motivation are not always sufficient to insure that two implementations of an option will be able to communicate. Therefore, a more complete description should be furnished in most cases. This description might take the form of text, a sample implementation, hints to implementers, etc.

A Note on "Subnegotiation"

Some options will require more information to be passed between hosts than a single option code. For example, any option which requires a parameter is such a case. The strategy to be used consists of two steps: first, both parties agree to "discuss" the parameter(s) and, second, the "discussion" takes place.

The first step, agreeing to discuss the parameters, takes place in the normal manner; one party proposes use of the option by sending a DO (or WILL) followed by the option code, and the other party accepts by returning a WILL (or DO) followed by the option code. Once both parties have agreed to use the option, subnegotiation takes place by using the command SB, followed by the option code, followed by the parameter(s), followed by the command SE. Each party is presumed to be able to parse the parameter(s), since each has indicated that the option is supported (via the initial exchange of WILL and DO). On the other hand, the receiver may locate the end of a parameter string by searching for the SE command (i.e., the string IAC SE), even if the receiver is unable to parse the parameters. Of course, either party may refuse to pursue further subnegotiation at any time by sending a WON'T or DON'T to the other party.

Thus, for option "ABC", which requires subnegotiation, the formats of the TELNET commands are:

IAC WILL ABC

Offer to use option ABC (or favorable acknowledgment of other party's request)

IAC DO ABC

Request for other party to use option ABC (or favorable acknowledgment of other party's offer)

IAC SB ABC <parameters> IAC SE

One step of subnegotiation, used by either party.

Designers of options requiring "subnegotiation" must take great care to avoid unending loops in the subnegotiation process. For example, if each party can accept any value of a parameter, and both parties suggest parameters with different values, then one is likely to have an infinite oscillation of "acknowledgments" (where each receiver believes it is only acknowledging the new proposals of the other). Finally, if parameters in an option "subnegotiation" include a byte with a value of 255, it is necessary to double this byte in accordance the general TELNET rules.

RFC-856 Telnet Binary Transmission

J. Postel & J. Reynolds
USC/Information Sciences Institute
May 1983

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet are expected to adopt and implement this standard.

Command Name and Code

TRANSMIT-BINARY 0

Default

WON'T TRANSMIT-BINARY
DON'T TRANSMIT-BINARY

The connection is not operated in binary mode.

RFC-856 Telnet Binary Option

Command Meanings

IAC WILL TRANSMIT-BINARY

The sender of this command REQUESTS permission to begin transmitting, or confirms that it will now begin transmitting characters which are to be interpreted as 8 bits of binary data by the receiver of the data.

IAC WON'T TRANSMIT-BINARY

If the connection is already being operated in binary transmission mode, the sender of this command DEMANDS to begin transmitting data characters which are to be interpreted as standard NVT ASCII characters by the receiver of the data. If the connection is not already being operated in binary transmission mode, the sender of this command REFUSES to begin transmitting characters which are to be interpreted as binary characters by the receiver of the data (i.e., the sender of the data demands to continue transmitting characters in its present mode).

A connection is being operated in binary transmission mode only when one party has requested it and the other has acknowledged it.

IAC DO TRANSMIT-BINARY

The sender of this command REQUESTS that the sender of the data start transmitting, or confirms that the sender of data is expected to transmit, characters which are to be interpreted as 8 bits of binary data (i.e., by the party sending this command).

IAC DON'T TRANSMIT-BINARY

If the connection is already being operated in binary transmission mode, the sender of this command DEMANDS that the sender of the data start transmitting characters which are to be interpreted as standard NVT ASCII characters by the receiver of the data (i.e., the party sending this command). If the connection is not already being operated in binary transmission mode, the sender of this command DEMANDS that the sender of data continue transmitting characters which are to be interpreted in the present mode.

A connection is being operated in binary transmission mode only when one party has requested it and the other has acknowledged it.

RFC-856 Telnet Binary Option

Motivation for the Option

It is sometimes useful to have available a binary transmission path within TELNET without having to utilize one of the more efficient, higher level protocols providing binary transmission (such as the File Transfer Protocol). The use of the IAC prefix within the basic TELNET protocol provides the option of binary transmission in a natural way, requiring only the addition of a mechanism by which the parties involved can agree to INTERPRET the characters transmitted over a TELNET connection as binary data.

RFC-856 Telnet Binary Option

Description of the Option

With the binary transmission option in effect, the receiver should interpret characters received from the transmitter which are not preceded with IAC as 8 bit binary data, with the exception of IAC followed by IAC which stands for the 8 bit binary data with the decimal value 255. IAC followed by an effective TELNET command (plus any additional characters required to complete the command) is still the command even with the binary transmission option in effect. IAC followed by a character which is not a defined TELNET command has the same meaning as IAC followed by NOP, although an IAC followed by an undefined command should not normally be sent in this mode.

Implementation Suggestions

It is foreseen that implementations of the binary transmission option will choose to refuse some other options (such as the EBCDIC transmission option) while the binary transmission option is in effect. However, if a pair of hosts can understand being in binary transmission mode simultaneous with being in, for example, echo mode, then it is all right if they negotiate that combination.

It should be mentioned that the meanings of WON'T and DON'T are dependent upon whether the connection is presently being operated in binary mode or not. Consider a connection operating in, say, EBCDIC mode which involves a system which has chosen not to implement any knowledge of the binary command. If this system were to receive a DO TRANSMIT-BINARY, it would not recognize the TRANSMIT-BINARY option and therefore would return a WON'T TRANSMIT-BINARY. If the default for the WON'T TRANSMIT-BINARY were always NVT ASCII, the sender of the DO TRANSMIT-BINARY would expect the recipient to have switched to NVT ASCII, whereas the receiver of the DO TRANSMIT-BINARY would not make this interpretation.

Thus, we have the rule that when a connection is not presently operating in binary mode, the default (i.e., the interpretation of WON'T and DON'T) is to continue operating in the current mode, whether that is NVT ASCII, EBCDIC, or some other mode. This rule, however, is not applied once a connection is operating in a binary mode (as agreed to by both ends); this would require each end of the connection to maintain a stack, containing all of the encoding-method transitions which had previously occurred on the connection, in order to properly interpret a WON'T or DON'T. Thus, a WON'T or DON'T received after the connection is operating in binary mode causes the encoding method to revert to NVT ASCII.

It should be remembered that a TELNET connection is a two way communication channel. The binary transmission mode must be negotiated separately for each direction of data flow, if that is desired.

Implementation of the binary transmission option, as is the case with implementations of all other TELNET options, must follow the loop preventing rules given in the General Considerations section of the TELNET Protocol Specification.

Consider now some issues of binary transmission both to and from both a process and a terminal:

a. Binary transmission from a terminal.

The implementer of the binary transmission option should consider how (or whether) a terminal transmitting over a TELNET connection with binary transmission in effect is allowed to generate all eight bit characters, ignoring parity considerations, etc., on input from the terminal.

b. Binary transmission to a process.

The implementer of the binary transmission option should consider how (or whether) all characters are passed to a process receiving over a connection with binary transmission in effect. As an example of the possible problem, TOPS-20 intercepts certain characters (e.g., ETX, the terminal control-C) at monitor level and does not pass them to the process.

c. Binary transmission from a process.

The implementer of the binary transmission option should consider how (or whether) a process transmitting over a connection with binary transmission in effect is allowed to send all eight bit characters with no characters intercepted

by the monitor and changed to other characters. An example of such a conversion may be found in the TOPS-20 system where certain non-printing characters are normally converted to a Circumflex (up-arrow) followed by a printing character.

d. Binary transmission to a terminal.

The implementer of the binary transmission option should consider how (or whether) all characters received over a connection with binary transmission in effect are sent to a local terminal. At issue may be the addition of timing characters normally inserted locally, parity calculations, and any normal code conversion.

RFC-857 Telnet Echo Option

J. Postel & J. Reynolds
USC/Information Sciences Institute
May 1983

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet are expected to adopt and implement this standard.

Command Name and Code

ECHO 1

Default

WON'T ECHO
DON'T ECHO

No echoing is done over the TELNET connection.

RFC-857 Telnet Echo Option

Command Meanings

IAC WILL ECHO

The sender of this command REQUESTS to begin, or confirms that it will now begin, echoing data characters it receives over the TELNET connection back to the sender of the data characters.

IAC WON'T ECHO

The sender of this command DEMANDS to stop, or refuses to start, echoing the data characters it receives over the TELNET connection back to the sender of the data characters.

IAC DO ECHO

The sender of this command REQUESTS that the receiver of this command begin echoing, or confirms that the receiver of this command is expected to echo, data characters it receives over the TELNET connection back to the sender.

IAC DON'T ECHO

The sender of this command DEMANDS the receiver of this command stop, or not start, echoing data characters it receives over the TELNET connection.

RFC-857 Telnet Echo Option

Motivation for the Option

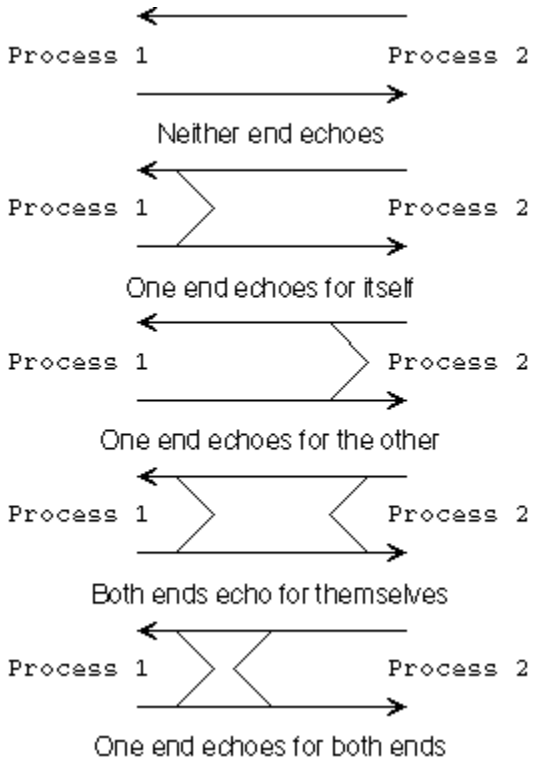
The NVT has a printer and a keyboard which are nominally interconnected so that "echoes" need never traverse the network; that is to say, the NVT nominally operates in a mode where characters typed on the keyboard are (by some means) locally turned around and printed on the printer. In highly interactive situations it is appropriate for the remote process (command language interpreter, etc.) to which the characters are being sent to control the way they are echoed on the printer. In order to support such interactive situations, it is necessary that there be a TELNET option to allow the parties at the two ends of the TELNET connection to agree that characters typed on an NVT keyboard are to be echoed by the party at the other end of the TELNET connection.

RFC-857 Telnet Echo Option

Description of the Option

When the echoing option is in effect, the party at the end performing the echoing is expected to transmit (echo) data characters it receives back to the sender of the data characters. The option does not require that the characters echoed be exactly the characters received (for example, a number of systems echo the ASCII ESC character with something other than the ESC character). When the echoing option is not in effect, the receiver of data characters should not echo them back to the sender; this, of course, does not prevent the receiver from responding to data characters received.

The normal TELNET connection is two way. That is, data flows in each direction on the connection independently; and neither, either, or both directions may be operating simultaneously in echo mode. There are five reasonable modes of operation for echoing on a connection pair:



This option provides the capability to decide on whether or not either end will echo for the other. It does not, however, provide any control over whether or not an end echoes for itself; this decision must be left to the sole discretion of the systems at each end (although they may use information regarding the state of "remote" echoing negotiations in making this decision).

It should be noted that if BOTH hosts enter the mode of echoing characters transmitted by the other host, then any character transmitted in either direction will be "echoed" back and forth indefinitely. Therefore, care should be taken in each implementation that if one site is echoing, echoing is not permitted to be turned on at the other.

As discussed in the TELNET Protocol Specification, both parties to a full-duplex TELNET connection initially assume each direction of the connection is being operated in the default mode which is non-echo (non-echo is not using this option, and the same as DON'T ECHO,

WON'T ECHO).

If either party desires himself to echo characters to the other party or for the other party to echo characters to him, that party gives the appropriate command (WILL ECHO or DO ECHO) and waits (and hopes) for acceptance of the option. If the request to operate the connection in echo mode is refused, then the connection continues to operate in non-echo mode. If the request to operate the connection in echo mode is accepted, the connection is operated in echo mode.

After a connection has been changed to echo mode, either party may demand that it revert to non-echo mode by giving the appropriate DON'T ECHO or WON'T ECHO command (which the other party must confirm thereby allowing the connection to operate in non-echo mode). Just as each direction of the TELNET connection may be put in remote echoing mode independently, each direction of the TELNET connection must be removed from remote echoing mode separately.

Implementations of the echo option, as implementations of all other TELNET options, must follow the loop preventing rules given in the General Considerations section of the TELNET Protocol Specification. Also, so that switches between echo and non-echo mode can be made with minimal confusion (momentary double echoing, etc.), switches in mode of operation should be made at times precisely coordinated with the reception and transmission of echo requests and demands. For instance, if one party responds to a DO ECHO with a WILL ECHO, all data characters received after the DO ECHO should be echoed and the WILL ECHO should immediately precede the first of the echoed characters.

The echoing option alone will normally not be sufficient to effect what is commonly understood to be remote computer echoing of characters typed on a terminal keyboard--the SUPPRESS-GO AHEAD option will normally have to be invoked in conjunction with the ECHO option to effect character-at-a-time remote echoing.

A Sample Implementation of the Option

The following is a description of a possible implementation for a simple user system called "UHOST".

A possible implementation could be that for each user terminal, the UHOST would keep three state bits: whether the terminal echoes for itself (UHOST ECHO always) or not (ECHO mode possible), whether the (human) user prefers to operate in ECHO mode or in non-ECHO mode, and whether the connection from this terminal to the server is in ECHO or non-ECHO mode. We will call these three bits P(hysical), D(esired), and A(ctual).

When a terminal dials up the UHOST the P-bit is set appropriately, the D-bit is set equal to it, and the A-bit is set to non-ECHO. The P-bit and D-bit may be manually reset by direct commands if the user so desires. For example, a user in Hawaii on a "full-duplex" terminal, would choose not to operate in ECHO mode, regardless of the preference of a mainland server. He should direct the UHOST to change his D-bit from ECHO to non-ECHO.

When a connection is opened from the UHOST terminal to a server, the UHOST would send the server a DO ECHO command if the MIN (with non-ECHO less than ECHO) of the P- and D-bits is different from the A-bit. If a WON'T ECHO or WILL ECHO arrives from the server, the UHOST will set the A-bit to the MIN of the received request, the P-bit, and the D-bit. If this changes the state of the A-bit, the UHOST will send off the appropriate acknowledgment; if it does not, then the UHOST will send off the appropriate refusal if not changing meant that it had to deny the request (i.e., the MIN of the P-and D-bits was less than the received A-request).

If while a connection is open, the UHOST terminal user changes either the P-bit or D-bit, the UHOST will repeat the above tests and send off a DO ECHO or DON'T ECHO, if necessary. When the connection is closed, the UHOST would reset the A-bit to indicate UHOST echoing.

While the UHOST's implementation would not involve DO ECHO or DON'T ECHO commands being sent to the server except when the connection is opened or the user explicitly changes his echoing mode, bigger hosts might invoke such mode switches quite frequently. For instance, while a line-at-a-time system were running, the server might attempt to put the user in local echo mode by sending the WON'T ECHO command to the user; but while a character-at-a-time system were running, the server might attempt to invoke remote echoing for the user by sending the WILL ECHO command to the user. Furthermore, while the UHOST will never send a WILL ECHO command and will only send a WON'T ECHO to refuse a server sent DO ECHO command, a server host might often send the WILL and WON'T ECHO commands.

RFC-858 Telnet Suppress Go-Ahead Option

J. Postel & J. Reynolds
USC/Information Sciences Institute
May 1983

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet are expected to adopt and implement this standard.

Command Name and Code

SUPPRESS-GO-AHEAD 3

Default

WON'T SUPPRESS-GO-AHEAD
DON'T SUPPRESS-GO-AHEAD

Go aheads are transmitted.

RFC-858 Telnet Suppress Go-Ahead Option

Command Meanings

IAC WILL SUPPRESS-GO-AHEAD

The sender of this command requests permission to begin suppressing transmission of the TELNET GO AHEAD (GA) character when transmitting data characters, or the sender of this command confirms it will now begin suppressing transmission of GAs with transmitted data characters.

IAC WON'T SUPPRESS-GO-AHEAD

The sender of this command demands to begin transmitting, or to continue transmitting, the GA character when transmitting data characters.

IAC DO SUPPRESS-GO-AHEAD

The sender of this commannd requests that the sender of data start suppressing GA when transmitting data, or the sender of this command confirms that the sender of data is expected to suppress transmission of GAs.

IAC DON'T SUPPRESSS-GO-AHEAD

The sender of this command demands that the receiver of the command start or continue transmitting GAs when transmitting data.

RFC-858 Telnet Suppress Go-Ahead Option

Motivation for the Option

While the NVT nominally follows a half duplex protocol complete with a GO AHEAD signal, there is no reason why a full duplex connection between a full duplex terminal and a host optimized to handle full duplex terminals should be burdened with the GO AHEAD signal. Therefore, it is desirable to have a TELNET option with which parties involved can agree that one or the other or both should suppress transmission of GO AHEADS.

RFC-858 Telnet Suppress Go-Ahead Option

Description of the Option

When the SUPPRESS-GO-AHEAD option is in effect on the connection between a sender of data and the receiver of the data, the sender need not transmit GAs.

It seems probable that the parties to the TELNET connection will suppress GO AHEAD in both directions of the TELNET connection if GO AHEAD is suppressed at all; but, nonetheless, it must be suppressed in both directions independently.

With the SUPPRESS-GO-AHEAD option in effect, the IAC GA command should be treated as a NOP if received, although IAC GA should not normally be sent in this mode.

Implementation Considerations

As the SUPRESS-GO-AHEAD option is sort of the opposite of a line at a time mode, the sender of data which is suppressing GO AHEADs should attempt to actually transmit characters as soon as possible (i.e., with minimal buffering) consistent with any other agreements which are in effect.

In many TELNET implementations it will be desirable to couple the SUPPRESS-GO-AHEAD option to the echo option so that when the echo option is in effect, the SUPPRESS-GO-AHEAD option is in effect simultaneously: both of these options will normally have to be in effect simultaneously to effect what is commonly understood to be character at a time echoing by the remote computer.

RFC-859 Telnet Status Option

J. Postel & J. Reynolds
USC/Information Sciences Institute
May 1983

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet are expected to adopt and implement this standard.

Command Name and Code

STATUS 5

Default

DON'T STATUS
WON'T STATUS

The current status of options will not be discussed.

Command Meanings

This option applies separately to each direction of data flow.

IAC DON'T STATUS

Sender refuses to carry on any further discussion of the current status of options.

IAC WON'T STATUS

Sender refuses to carry on any further discussion of the current status of options.

IAC SB STATUS SEND IAC SE

Sender requests receiver to transmit his (the receiver's) perception of the current status of Telnet options. The code for SEND is 1.

IAC SB STATUS IS ... IAC SE

Sender is stating his perception of the current status of Telnet options. The code for IS is 0.

RFC-859 Telnet Status Option

Motivation for the Option

This option allows a user/process to verify the current status of TELNET options (e.g., echoing) as viewed by the person/process on the other end of the TELNET connection. Simply renegotiating options could lead to the nonterminating request loop problem discussed in the General Consideration section of the TELNET Specification. This option fits into the normal structure of TELNET options by deferring the actual transfer of status information to the SB command.

RFC-859 Telnet Status Option

Description of the Option

WILL and DO are used only to obtain and grant permission for future discussion. The actual exchange of status information occurs within option subcommands (IAC SB STATUS...).

Once the two hosts have exchanged a WILL and a DO, the sender of the WILL STATUS is free to transmit status information, spontaneously or in response to a request from the sender of the DO. At worst, this may lead to transmitting the information twice. Only the sender of the DO may send requests (IAC SB STATUS SEND IAC SE) and only the sender of the WILL may transmit actual status information (within an IAC SB STATUS IS ... IAC SE command).

IS has the subcommands WILL, DO and SB. They are used EXACTLY as used during the actual negotiation of TELNET options, except that SB is terminated with SE, rather than IAC SE. Transmission of SE, as a regular data byte, is accomplished by doubling the byte (SE SE). Options that are not explicitly described are assumed to be in their default states. A single IAC SB STATUS IS ... IAC SE describes the condition of ALL options.

Example

Host1: IAC DO STATUS

Host2: IAC WILL STATUS

(Host2 is now free to send status information at any time. Solicitations from Host1 are NOT necessary. This should not produce any dangerous race conditions. At worst, two IS's will be sent.)

Host1 (perhaps): IAC SB STATUS SEND IAC SE

Host2 (the following stream is broken into multiple lines only for readability. No carriage returns are implied.):

IAC SB STATUS IS
WILL ECHO
DO SUPPRESS-GO-AHEAD
WILL STATUS
DO STATUS
IAC SE

Explanation of Host2's perceptions
It is responsible for echoing back the data characters it receives over the TELNET connection; it will not send Go-Ahead signals; it will both issue and request Status information.

RFC-860 Telnet Timing Mark Option

J. Postel & J. Reynolds
USC/Information Sciences Institute
May 1983

This RFC specifies a standard for the ARPA community. Hosts on the ARPA Internet are expected to adopt and implement this standard.

Command Name and Code

TIMING-MARK 6

Default

WON'T TIMING-MARK
DON'T TIMING-MARK

No explicit attempt is made to synchronize the activities at the two ends of the TELNET connection.

Command Meanings

IAC DO TIMING-MARK

The sender of this command REQUESTS that the receiver of this command return a WILL TIMING-MARK in the data stream at the "appropriate place" as defined in Motivation.

IAC WILL TIMING-MARK

The sender of this command ASSURES the receiver of this command that it is inserted in the data stream at the "appropriate place" to insure synchronization with a DO TIMING-MARK transmitted by the receiver of this command.

IAC WON'T TIMING-MARK

The sender of this command REFUSES to insure that this command is inserted in the data stream at the "appropriate place" to insure synchronization.

IAC DON'T TIMING-MARK

The sender of this command notifies the receiver of this command that a WILL TIMING-MARK (previously transmitted by the receiver of this command) has been IGNORED.

RFC-860 Telnet Timing Mark Option

Motivation for the Option

It is sometimes useful for a user or process at one end of a TELNET connection to be sure that previously transmitted data has been completely processed, printed, discarded, or otherwise disposed of. This option provides a mechanism for doing this. In addition, even if the option request (DO TIMING-MARK) is refused (by WON'T TIMING-MARK) the requester is at least assured that the refuser has received (if not processed) all previous data.

As an example of a particular application, imagine a TELNET connection between a physically full duplex terminal and a "full duplex" server system which permits the user to "type ahead" while the server is processing previous user input. Suppose that both sides have agreed to Suppress Go Ahead and that the server has agreed to provide echoes. The server now discovers a command which it cannot parse, perhaps because of a user typing error. It would like to throw away all of the user's "type-ahead" (since failure of the parsing of one command is likely to lead to incorrect results if subsequent commands are executed), send the user an error message, and resume interpretation of commands which the user typed after seeing the error message. If the user were local, the system would be able to discard the buffered input; but input may be buffered in the user's host or elsewhere. Therefore, the server might send a DO TIMING-MARK and hope to receive a WILL TIMING-MARK from the user at the "appropriate place" in the data stream.

The "appropriate place", therefore (in absence of other information) is clearly just before the first character which the user typed after seeing the error message. That is, it should appear that the timing mark was "printed" on the user's terminal and that, in response, the user typed an answering timing mark.

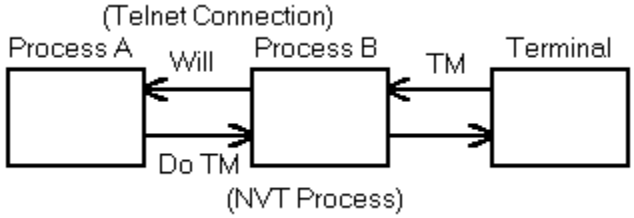
Next, suppose that the user in the example above realized that he had misspelled a command, realized that the server would send a DO TIMING-MARK, and wanted to start "typing ahead" again without waiting for this to occur. He might then instruct his own system to send a WILL TIMING-MARK to the server and then begin "typing ahead" again. (Implementers should remember that the user's own system must remember that it sent the WILL TIMING-MARK so as to discard the DO/DON'T TIMING-MARK when it eventually arrives.) Thus, in this case the "appropriate place" for the insertion of the WILL TIMING-MARK is the place defined by the user.

It should be noted, in both of the examples above, that it is the responsibility of the system which transmits the DO TIMING-MARK to discard any unwanted characters; the WILL TIMING-MARK only provides help in deciding which characters are "unwanted".

RFC-860 Telnet Timing Mark Option

Description of the Option

Suppose that Process A of the figure shown below wishes to synchronize with B. The DO TIMING-MARK is sent from A to B. B can refuse by replying WON'T TIMING-MARK, or agree by permitting the timing mark to flow through his "outgoing" buffer, BUF2. Then, instead of delivering it to the terminal, B will enter the mark into his "incoming" buffer BUF1, to flow through toward A. When the mark has propagated through B's incoming buffer, B returns the WILL TIMING-MARK over the TELNET connection to A.



When A receives the WILL TIMING-MARK, he knows that all the information he sent to B before sending the timing mark been delivered, and all the information sent from B to A before turnaround of the timing mark has been delivered.

Three typical applications are:

A. Measure round-trip delay between a process and a terminal or another process.

B. Resynchronizing an interaction as described in section 4 above. A is a process interpreting commands forwarded from a terminal by B. When A sees an illegal command it:

   i. Sends <carriage return>, <line feed>, <question mark>.

   ii. Sends DO TIMING-MARK.

   iii. Sends an error message.

   iv. Starts reading input and throwing it away until it receives a WILL TIMING-MARK.

   v. Resumes interpretation of input.

   This achieves the effect of flushing all "type ahead" after the erroneous command, up to the point when the user actually saw the question mark.

C. The dual of B above. The terminal user wants to throw away unwanted output from A.

   i. B sends DO TIMING-MARK, followed by some new command.

   ii. B starts reading output from A and throwing it away until it receives WILL TIMING-MARK.

   iii. B resumes forwarding A's output to the terminal.

   This achieves the effect of flushing all output from A, up to the point where A saw the timing mark, but not output generated in response to the following command.

RFC-861 Telnet Extended Options-List Option

J. Postel & J. Reynolds
USC/Information Sciences Institute
May 1983

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet are expected to adopt and implement this standard.

Command Name and Code

EXTENDED-OPTIONS-LIST (EXOPL) 255

Default

WON'T EXOPL
DON'T EXOPL

Negotiation of options on the "Extended Options List" is not permitted.

Command Meanings

IAC DO EXOPL

The sender of this command REQUESTS that the receiver of this command begin negotiating, or confirms that the receiver of this command is expected to begin negotiating, TELNET options which are on the "Extended Options List".

IAC WILL EXOPL

The sender of this command requests permission to begin negotiating, or confirms that it will begin negotiating, TELNET options which are on the "Extended Options List".

IAC WON'T EXOPL

The sender of this command REFUSES to negotiate, or to continue negotiating, options on the "Extended Options List".

IAC DON'T EXOPL

The sender of this command DEMANDS that the receiver conduct no further negotiation of options on the "Extended Options List".

IAC SB EXOPL <subcommand>

The subcommand contains information required for the negotiation of an option of the "Extended Options List". The format of the subcommand is discussed in the description below.

RFC-861 Telnet Extended Options-List Option

Motivation for the Option

Eventually, a 257th TELNET option will be needed. This option will extend the option list for another 256 options in a manner which is easy to implement. The option is proposed now, rather than later (probably much later), in order to reserve the option number (255).

An Abstract Description of the Option

The EXOPL option has five subcommand codes: WILL, WON'T, DO, DON'T, and SB. They have exactly the same meanings as the TELNET commands with the same names, and are used in exactly the same way. For consistency, these subcommand codes will have the same values as the TELNET command codes (250-254). Thus, the format for negotiating a specific option on the "Extended Options List" (once both parties have agreed to use it) is:

IAC SB EXOPL DO/DON'T/WILL/WON'T/<option code> IAC SE

Once both sides have agreed to use the specific option specified by <option code>, subnegotiation may be required. In this case the format to be used is:

IAC SB EXOPL SB <option code> <parameters> SE IAC SE

RFC-862 Echo Protocol

J. Postel
USC/Information Sciences Institute
May 1983

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet that choose to implement an Echo Protocol are expected to adopt and implement this standard.

A very useful debugging and measurement tool is an echo service. An echo service simply sends back to the originating source any data it receives.

TCP Based Echo Service

One echo service is defined as a connection based application on TCP. A server listens for TCP connections on TCP port 7. Once a connection is established any data received is sent back. This continues until the calling user terminates the connection.

UDP Based Echo Service

Another echo service is defined as a datagram based application on UDP. A server listens for UDP datagrams on UDP port 7. When a datagram is received, the data from it is sent back in an answering datagram.

RFC-863 Discard Protocol

J. Postel
USC/Information Sciences Institute
May 1983

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet that choose to implement a Discard Protocol are expected to adopt and implement this standard.

A useful debugging and measurement tool is a discard service. A discard service simply throws away any data it receives.

TCP Based Discard Service

One discard service is defined as a connection based application on TCP. A server listens for TCP connections on TCP port 9. Once a connection is established any data received is thrown away. No response is sent. This continues until the calling user terminates the connection.

UDP Based Discard Service

Another discard service is defined as a datagram based application on UDP. A server listens for UDP datagrams on UDP port 9. When a datagram is received, it is thrown away. No response is sent.

RFC-864 Character Generator Protocol

J. Postel
USC/Information Sciences Institute
May 1983

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet that choose to implement a Character Generator Protocol are expected to adopt and implement this standard.

A useful debugging and measurement tool is a character generator service. A character generator service simply sends data without regard to the input.

TCP Based Character Generator Service

One character generator service is defined as a connection based application on TCP. A server listens for TCP connections on TCP port 19. Once a connection is established a stream of data is sent out the connection (and any data received is thrown away). This continues until the calling user terminates the connection.

It is fairly likely that users of this service will abruptly decide that they have had enough and abort the TCP connection, instead of carefully closing it. The service should be prepared for either the carfull close or the rude abort.

The data flow over the connection is limited by the normal TCP flow control mechanisms, so there is no concern about the service sending data faster than the user can process it.

UDP Based Character Generator Service

Another character generator service is defined as a datagram based application on UDP. A server listens for UDP datagrams on UDP port 19. When a datagram is received, an answering datagram is sent containing a random number (between 0 and 512) of characters (the data in the received datagram is ignored).

There is no history or state information associated with the UDP version of this service, so there is no continuity of data from one answering datagram to another.

The service only send one datagram in response to each received datagram, so there is no concern about the service sending data faster than the user can process it.

Data Syntax

The data may be anything. It is recommended that a recognizable pattern be used in tha data.

> One popular pattern is 72 chraracter lines of the ASCII printing characters. There are 95 printing characters in the ASCII character set. Sort the characters into an ordered sequence and number the characters from 0 through 94. Think of the sequence as a ring so that character number 0 follows character number 94. On the first line (line 0) put the characters numbered 0 through 71. On the next line (line 1) put the characters numbered 1 through 72. And so on. On line N, put characters (0+N mod 95) through (71+N mod 95). End each line with carriage return and line feed.

RFC-864 Character Generator Protocol

Example

```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefgh
"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghi
#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghij
$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijk
%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijkl
&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklm
'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmn
()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmno
)*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnop
*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopq
+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqr
,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrs
-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrst
./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstu
/0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuv
0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvw
123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwx
23456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxy
3456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz
456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{
56789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|
6789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}
789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
89:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
9:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !
:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"
;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#
<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$
=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%
>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&
?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'(
ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'()
BCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'()*
CDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'()*+
DEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'()*+,
EFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'()*+,-
FGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'()*+,-.
GHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'()*+,-./
HIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'()*+,-./0
IJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'()*+,-./01
JKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'()*+,-./012
KLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'()*+,-./0123
LMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'()*+,-./01234
MNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'()*+,-./012345
NOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'()*+,-./0123456
```

RFC-865 Quote of the Day Protocol

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet that choose to implement a Quote of the Day Protocol are expected to adopt and implement this standard.

A useful debugging and measurement tool is a quote of the day service. A quote of the day service simply sends a short message without regard to the input.

TCP Based Quote of the Day Service

One quote of the day service is defined as a connection based application on TCP. A server listens for TCP connections on TCP port 17. Once a connection is established a short message is sent out the connection (and any data received is thrown away). The service closes the connection after sending the quote.

UDP Based Quote of the Day Service

Another quote of the day service is defined as a datagram based application on UDP. A server listens for UDP datagrams on UDP port 17. When a datagram is received, an answering datagram is sent containing a quote (the data in the received datagram is ignored).

Quote Syntax

There is no specific syntax for the quote. It is recommended that it be limited to the ASCII printing characters, space, carriage return, and line feed. The quote may be just one or up to several lines, but it should be less than 512 characters.

RFC-866 Active Users Protocol

J. Postel
USC/Information Sciences Institute
May 1983

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet that choose to implement an Active Users Protocol are expected to adopt and implement this standard.

A useful debugging and measurement tool is an active users service. An active users service simply sends a list of the currently active users on the host without regard to the input.

An active user is one logged in, such as listed in SYSTAT or WHO.

TCP Based Active Users Service

One active users service is defined as a connection based application on TCP. A server listens for TCP connections on TCP port 11. Once a connection is established a list of the currently active users is sent out the connection (and any data received is thrown away). The service closes the connection after sending the list.

UDP Based Active Users Service

Another active users service service is defined as a datagram based application on UDP. A server listens for UDP datagrams on UDP port 11. When a datagram is received, an answering datagram is sent containing a list of the currently active users (the data in the received datagram is ignored).

If the list does not fit in one datagram then send a sequence of datagrams but don't break the information for a user (a line) across a datagram. The user side should wait for a timeout for all datagrams to arrive. Note that they are not necessarily in order.

User List Syntax

There is no specific syntax for the user list. It is recommended that it be limited to the ASCII printing characters, space, carriage return, and line feed. Each user should be listed on a separate line.

RFC-867 Daytime Protocol

J. Postel
USC/Information Sciences Institute
May 1983

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet that choose to implement a Daytime Protocol are expected to adopt and implement this standard.

A useful debugging and measurement tool is a daytime service. A daytime service simply sends a the current date and time as a character string without regard to the input.

TCP Based Daytime Service

One daytime service is defined as a connection based application on TCP. A server listens for TCP connections on TCP port 13. Once a connection is established the current date and time is sent out the connection as a ascii character string (and any data received is thrown away). The service closes the connection after sending the string.

UDP Based Daytime Service

Another daytime service service is defined as a datagram based application on UDP. A server listens for UDP datagrams on UDP port 13. When a datagram is received, an answering datagram is sent containing the current date and time as a ASCII character string (the data in the received datagram is ignored).

Daytime Syntax

There is no specific syntax for the daytime. It is recommended that it be limited to the ASCII printing characters, space, carriage return, and line feed. The daytime should be just one line.

One popular syntax is:

Weekday, Month Day, Year Time-Zone

Example:

Tuesday, February 22, 1982 17:37:43-PST

Another popular syntax is that used in SMTP:

dd mmm yy hh:mm:ss zzz

Example:

02 FEB 82 07:59:01 PST

NOTE: For machine useful time use the Time Protocol (RFC-868) or the Network Time Protocol (RFC-1119).

RFC-868 Time Protocol

J. Postel - USC/ISI
K. Harrenstien - SRI
May 1983

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet that choose to implement a Time Protocol are expected to adopt and implement this standard.

This protocol provides a site-independent, machine readable date and time. The Time service sends back to the originating source the time in seconds since midnight on January first 1900.

One motivation arises from the fact that not all systems have a date/time clock, and all are subject to occasional human or machine error. The use of time-servers makes it possible to quickly confirm or correct a system's idea of the time, by making a brief poll of several independent sites on the network.

This protocol may be used either above the Transmission Control Protocol (TCP) or above the User Datagram Protocol (UDP).

When used via TCP the time service works as follows:

S: Listen on port 37 (45 octal).
U: Connect to port 37.
S: Send the time as a 32 bit binary number.
U: Receive the time.
U: Close the connection.
S: Close the connection.

The server listens for a connection on port 37. When the connection is established, the server returns a 32-bit time value and closes the connection. If the server is unable to determine the time at its site, it should either refuse the connection or close it without sending anything.

When used via UDP the time service works as follows:

S: Listen on port 37 (45 octal).
U: Send an empty datagram to port 37.
S: Receive the empty datagram.
S: Send a datagram containing the time as a 32 bit binary number.
U: Receive the time datagram.

The server listens for a datagram on port 37. When a datagram arrives, the server returns a datagram containing the 32-bit time value. If the server is unable to determine the time at its site, it should discard the arriving datagram and make no reply.

The Time

The time is the number of seconds since 00:00 (midnight) 1 January 1900 GMT, such that the time 1 is 12:00:01 am on 1 January 1900 GMT; this base will serve until the year 2036.

For example:

the time 2,208,988,800 corresponds to 00:00 1 Jan 1970 GMT,
2,398,291,200 corresponds to 00:00 1 Jan 1976 GMT,
2,524,521,600 corresponds to 00:00 1 Jan 1980 GMT,
2,629,584,000 corresponds to 00:00 1 May 1983 GMT,
and -1,297,728,000 corresponds to 00:00 17 Nov 1858 GMT.

RFC-879

The TCP Maximum Segment Size Option

and Related Topics

J. Postel
USC/Information Sciences Institute
November 1983

This memo discusses the TCP Maximum Segment Size Option and related topics. The purposes is to clarify some aspects of TCP and its interaction with IP. This memo is a clarification to the TCP specification, and contains information that may be considered as "advice to implementers".

RFC-879 The TCP Maximum Segment Size Option and Related Topics

Introduction

This memo discusses the TCP Maximum Segment Size and its relation to the IP Maximum Datagram Size. TCP is specified in RFC-793. IP and ICMP are specified in RFC-791 and RFC-792.

This discussion is necessary because the current specification of this TCP option is ambiguous.

Much of the difficulty with understanding these sizes and their relationship has been due to the variable size of the IP and TCP headers.

There have been some assumptions made about using other than the default size for datagrams with some unfortunate results.

> **Hosts must not send datagrams larger than 576 octets unless they have specific knowledge that the destination host is prepapred to accept larger datagrams.**
>
> This is a long established rule.

To resolve the ambiguity in the TCP Maximum Segment Size option definition the following rule is established:

> **The TCP maximum segment size is the IP maximum datagram size minus forty.**
>
> The default IP Maximum Datagram Size is 576.
>
> The default TCP Maximum Segment Size is 536.

The IP Maximum Datagram Size

Hosts are not required to reassemble infinitely large IP datagrams. The maximum size datagram that all hosts are required to accept or reassemble from fragments is 576 octets. The maximum size reassembly buffer every host must have is 576 octets. Hosts are allowed to accept larger datagrams and assemble fragments into larger datagrams, hosts may have buffers as large as they please.

Hosts must not send datagrams larger than 576 octets unless they have specific knowledge that the destination host is prepared to accept larger datagrams.

The TCP Maximum Segment Size Option

TCP provides an option that may be used at the time a connection is established (only) to indicate the maximum size TCP segment that can be accepted on that connection. This Maximum Segment Size (MSS) announcement (often mistakenly called a negotiation) is sent from the data receiver to the data sender and says "I can accept TCP segments up to size X". The size (X) may be larger or smaller than the default. The MSS can be used completely independently in each direction of data flow. The result may be quite different maximum sizes in the two directions.

The MSS counts only data octets in the segment, it does not count the TCP header or the IP header.

A footnote
The MSS value counts only data octets, thus it does not count the TCP SYN and FIN control bits even though SYN and FIN do consume TCP sequence numbers.

The Relationship of TCP Segments and IP Datagrams

TCP segment are transmitted as the data in IP datagrams. The correspondence between TCP segments and IP datagrams must be one to one. This is because TCP expects to find exactly one complete TCP segment in each block of data turned over to it by IP, and IP must turn over a block of data for each datagram received (or completely reassembled).

RFC-879 The TCP Maximum Segment Size Option and Related Topics

Layering and Modularity

TCP is an end to end reliable data stream protocol with error control, flow control, etc. TCP remembers many things about the state of a connection.

IP is a one shot datagram protocol. IP has no memory of the datagrams transmitted. It is not appropriate for IP to keep any information about the maximum datagram size a particular destination host might be capable of accepting.

TCP and IP are distinct layers in the protocol architecture, and are often implemented in distinct program modules.

Some people seem to think that there must be no communication between protocol layers or program modules. There must be communication between layers and modules, but it should be carefully specified and controlled. One problem in understanding the correct view of communication between protocol layers or program modules in general, or between TCP and IP in particular is that the documents on protocols are not very clear about it. This is often because the documents are about the protocol exchanges between machines, not the program architecture within a machine, and the desire to allow many program architectures with different organization of tasks into modules.

IP Information Requirements

There is no general requirement that IP keep information on a per host basis.

IP must make a decision about which directly attached network address to send each datagram to. This is simply mapping an IP address into a directly attached network address.

There are two cases to consider: the destination is on the same network, and the destination is on a different network.

Same Network
For some networks the the directly attached network address can be computed from the IP address for destination hosts on the directly attached network.

For other networks the mapping must be done by table look up (however the table is initialized and maintained, for example when using the Address Resolution Protocol).

Different Network
The IP address must be mapped to the directly attached network address of a gateway. For networks with one gateway to the rest of the Internet the host need only determine and remember the gateway address and use it for sending all datagrams to other networks.

For networks with multiple gateways to the rest of the Internet, the host must decide which gateway to use for each datagram sent. It need only check the destination network of the IP address and keep information on which gateway to use for each network.

The IP does, in some cases, keep per host routing information for other hosts on the directly attached network. The IP does, in some cases, keep per network routing information.

A Special Case
There are two ICMP messages that convey information about particular hosts. These are subtypes of the Destination Unreachable and the Redirect ICMP messages. These messages are expected only in very unusual circumstances. To make effective use of these messages the receiving host would have to keep information about the specific hosts reported on. Because these messages are quite rare it is strongly recommended that this be done through an exception mechanism rather than having the IP keep per host tables for all hosts.

RFC-879 The TCP Maximum Segment Size Option and Related Topics

The Relationship between IP Datagram and TCP Segment Sizes

The relationship between the value of the maximum IP datagram size and the maximum TCP segment size is obscure. The problem is that both the IP header and the TCP header may vary in length. The TCP Maximum Segment Size option (MSS) is defined to specify the maximum number of data octets in a TCP segment exclusive of TCP (or IP) header.

To notify the data sender of the largest TCP segment it is possible to receive the calculation of the MSS value to send is:

MSS = MTU - sizeof(TCPHDR) - sizeof(IPHDR)

On receipt of the MSS option the calculation of the size of segment that can be sent is:

SndMaxSegSiz = MIN((MTU - sizeof(TCPHDR) - sizeof(IPHDR)), MSS)

where MSS is the value in the option, and MTU is the Maximum Transmission Unit (or the maximum packet size) allowed on the directly attached network.

This begs the question, though. What value should be used for the "sizeof(TCPHDR)" and for the "sizeof(IPHDR)"?

There are three reasonable positions to take: the conservative, the moderate, and the liberal.

1. The conservative or pessimistic position assumes the worst -- that both the IP header and the TCP header are maximum size, that is, 60 octets each.

   MSS = MTU - 60 - 60 = MTU - 120

   If MTU is 576 then MSS = 456

2. The moderate position assumes the that the IP is maximum size (60 octets) and the TCP header is minimum size (20 octets), because there are no TCP header options currently defined that would normally be sent at the same time as data segments.

   MSS = MTU - 60 - 20 = MTU - 80

   If MTU is 576 then MSS = 496

3. The liberal or optimistic position assumes the best -- that both the IP header and the TCP header are minimum size, that is, 20 octets each.

   MSS = MTU - 20 - 20 = MTU - 40

   If MTU is 576 then MSS = 536

If nothing is said about MSS, the data sender may cram as much as possible into a 576 octet datagram, and if the datagram has minimum headers (which is most likely), the result will be 536 data octets in the TCP segment. The rule relating MSS to the maximum datagram size ought to be consistent with this.

A practical point is raised in favor of the liberal position too. Since the use of minimum IP and TCP headers is very likely in the very large percentage of cases, it seems wasteful to limit the TCP segment data to so much less than could be transmitted at once, especially since it is less that 512 octets.

For comparison: 536/576 is 93% data, 496/576 is 86% data, 456/576 is 79% data.

RFC-879 The TCP Maximum Segment Size Option and Related Topics

Maximum Packet Size

Each network has some maximum packet size, or maximum transmission unit (MTU). Ultimately there is some limit imposed by the technology, but often the limit is an engineering choice or even an administrative choice. Different installations of the same network product do not have to use the same maximum packet size. Even within one installation not all host must use the same packet size (this way lies madness, though).

Some IP implementers have assumed that all hosts on the directly attached network will be the same or at least run the same implementation. This is a dangerous assumption. It has often developed that after a small homogeneous set of host have become operational additional hosts of different types are introduced into the environment. And it has often developed that it is desired to use a copy of the implementation in a different inhomogeneous environment.

Designers of gateways should be prepared for the fact that successful gateways will be copied and used in other situation and installations. Gateways must be prepared to accept datagrams as large as can be sent in the maximum packets of the directly attached networks. Gateway implementations should be easily configured for installation in different circumstances.

A footnote:
The MTUs of some popular networks (note that the actual limit in some installations may be set lower by administrative policy):

ARPANET, MILNET = 1007
Ethernet (10Mb) = 1500
Proteon PRONET = 2046

Source Fragmentation

A source host would not normally create datagram fragments. Under normal circumstances datagram fragments only arise when a gateway must send a datagram into a network with a smaller maximum packet size than the datagram. In this case the gateway must fragment the datagram (unless it is marked "don't fragment" in which case it is discarded, with the option of sending an ICMP message to the source reporting the problem).

It might be desirable for the source host to send datagram fragments if the maximum segment size (default or negotiated) allowed by the data receiver were larger than the maximum packet size allowed by the directly attached network. However, such datagram fragments must not combine to a size larger than allowed by the destination host.

> For example, if the receiving TCP announced that it would accept segments up to 5000 octets (in cooperation with the receiving IP) then the sending TCP could give such a large segment to the sending IP provided the sending IP would send it in datagram fragments that fit in the packets of the directly attached network.

There are some conditions where source host fragmentation would be necessary.

> If the host is attached to a network with a small packet size (for example 256 octets), and it supports an application defined to send fixed sized messages larger than that packet size (for example TFTP).
>
> If the host receives ICMP Echo messages with data it is required to send an ICMP Echo-Reply message with the same data. If the amount of data in the Echo were larger than the packet size of the directly attached network the following steps might be required: (1) receive the fragments, (2) reassemble the datagram, (3) interpret the Echo, (4) create an Echo-Reply, (5) fragment it, and (6) send the fragments.

Gateway Fragmentation

Gateways must be prepared to do fragmentation. It is not an optional feature for a gateway.

Gateways have no information about the size of datagrams destination hosts are prepared to accept. It would be inappropriate for gateways to attempt to keep such information.

Gateways must be prepared to accept the largest datagrams that are allowed on each of the directly attached networks, even if it is larger than 576 octets.

Gateways must be prepared to fragment datagrams to fit into the packets of the next network, even if it smaller than 576 octets.

If a source host thought to take advantage of the local network's ability to carry larger datagrams but doesn't have the slightest idea if the destination host can accept larger than default datagrams and expects the gateway to fragment the datagram into default size fragments, then the source host is misguided. If indeed, the destination host can't accept larger than default datagrams, it probably can't reassemble them either. If the gateway either passes on the large datagram whole or fragments into default size fragments the destination will not accept it. Thus, this mode of behavior by source hosts must be outlawed.

A larger than default datagram can only arrive at a gateway because the source host knows that the destination host can handle such large datagrams (probably because the destination host announced it to the source host in an TCP MSS option). Thus, the gateway should pass on this large datagram in one piece or in the largest fragments that fit into the next network.

An interesting footnote is that even though the gateways may know about know the 576 rule, it is irrelevant to them.

RFC-879 The TCP Maximum Segment Size Option and Related Topics

Inter-Layer Communication

The Network Driver (ND) or interface should know the Maximum Transmission Unit (MTU) of the directly attached network.

The IP should ask the Network Driver for the Maximum Transmission Unit.

The TCP should ask the IP for the Maximum Datagram Data Size (MDDS). This is the MTU minus the IP header length (MDDS = MTU - IPHdrLen).

When opening a connection TCP can send an MSS option with the value equal MDDS - TCPHdrLen.

TCP should determine the Maximum Segment Data Size (MSDS) from either the default or the received value of the MSS option.

TCP should determine if source fragmentation is possible (by asking the IP) and desirable.

> If so TCP may hand to IP segments (including the TCP header) up to MSDS + TCPHdrLen.
>
> If not TCP may hand to IP segments (including the TCP header) up to the lesser of (MSDS + TCPHdrLen) and MDDS.

IP checks the length of data passed to it by TCP. If the length is less than or equal MDDS, IP attached the IP header and hands it to the ND. Otherwise the IP must do source fragmentation.

What is the Default MSS ?

Another way of asking this question is "What transmitted value for MSS has exactly the same effect of not transmitting the option at all?".

In terms of the previous section:

> The default assumption is that the Maximum Transmission Unit is 576 octets.
>
> > MTU = 576
>
> The Maximum Datagram Data Size (MDDS) is the MTU minus the IP header length.
>
> > MDDS = MTU - IPHdrLen = 576 - 20 = 556

When opening a connection TCP can send an MSS option with the value equal MDDS - TCPHdrLen.

> MSS = MDDS - TCPHdrLen = 556 - 20 = 536

TCP should determine the Maximum Segment Data Size (MSDS) from either the default or the received value of the MSS option.

> Default MSS = 536, then MSDS = 536

TCP should determine if source fragmentation is possible and desirable.

If so TCP may hand to IP segments (including the TCP header) up to MSDS + TCPHdrLen (536 + 20 = 556).

If not TCP may hand to IP segments (including the TCP header) up to the lesser of (MSDS + TCPHdrLen (536 + 20 = 556)) and MDDS (556).

RFC-879 The TCP Maximum Segment Size Option and Related Topics

The Truth

The rule relating the maximum IP datagram size and the maximum TCP segment size is:

> TCP Maximum Segment Size = IP Maximum Datagram Size - 40

The rule must match the default case.

> If the TCP Maximum Segment Size option is not transmitted then the data sender is allowed to send IP datagrams of maximum size (576) with a minimum IP header (20) and a minimum TCP header (20) and thereby be able to stuff 536 octets of data into each TCP segment.

The definition of the MSS option can be stated:

> The maximum number of data octets that may be received by the sender of this TCP option in TCP segments with no TCP header options transmitted in IP datagrams with no IP header options.

The Consequences

When TCP is used in a situation when either the IP or TCP headers are not minimum and yet the maximum IP datagram that can be received remains 576 octets then the TCP Maximum Segment Size option must be used to reduce the limit on data octets allowed in a TCP segment.

> For example, if the IP Security option (11 octets) were in use and the IP maximum datagram size remained at 576 octets, then the TCP should send the MSS with a value of 525 (536-11).

RFC-885 Telnet End-of-Record Option

J. Postel
USC/Information Sciences Institute
December 1983

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet that need to mark record boundaries within Telnet protocol data are expected to adopt and implement this standard.

Command Name and Code

END-OF-RECORD 25

Default

WON'T END-OF-RECORD
DON'T END-OF-RECORD

End-Of-Record is not transmitted.

Command Meanings

IAC WILL END-OF-RECORD

The sender of this command requests permission to begin transmission of the Telnet END-OF-RECORD (EOR) code when transmitting data characters, or the sender of this command confirms it will now begin transmission of EORs with transmitted data characters.

IAC WON'T END-OF-RECORD

The sender of this command demands to stop transmitting, or to refuses to begin transmitting, the EOR code when transmitting data characters.

IAC DO END-OF-RECORD

The sender of this command requests that the sender of data start transmitting the EOR code when transmitting data, or the sender of this command confirms that the sender of data is expected to transmit EORs.

IAC DON'T END-OF-RECORD

The sender of this command demands that the receiver of the command stop or not start transmitting EORs when transmitting data.

RFC-885 Telnet End-of-Record Option

Motivation for the Option

Many interactive systems use one (or more) of the normal data characters to indicate the end of an effective unit of data (i.e., a record), for example, carriage-return (or line-feed, or escape). Some systems, however, have some special means of indicating the end of an effective data unit, for example, a special key. This Telnet option provides a means of communicating the end of data unit in a standard way.

RFC-885 Telnet End-of-Record Option

Description of the Option

When the END-OF-RECORD option is in effect on the connection between a sender of data and the receiver of the data, the sender transmits EORs.

It seems probable that the parties to the Telnet connection will transmit EORs in both directions of the Telnet connection if EORs are used at all; however, the use of EORs must be negotiated independently for each direction.

When the END-OF-RECORD option is not in effect, the IAC EOR command should be treated as a NOP if received, although IAC EOR should not normally be sent in this mode.

Implementation Considerations

As the EOR code indicates the end of an effective data unit, Telnet should attempt to send the data up to and including the EOR code together to promote communication efficiency.

The end of record is indicated by the IAC EOR 2-octet sequence. The code for EOR is 239 (decimal).

RFC-893 Trailer Encapsulations

Samuel J. Leffler & Michael J. Karels
University of California at Berkeley
April 1984

Status of this Memo

When this RFC was originally published in April of 1984 it was for informational purposes only. It has since become an official protocol of the Internet community, although its use is **optional**. It may only be used when it has been verified that both systems (host or gateway) involved in the link-layer communication implement trailers. If a system does not dynamically negotiate use of the trailer protocol on a per-destination basis, the default configuration **must** disable the protocol.

RFC-893 Trailer Encapsulations

Introduction

A trailer encapsulation is a link level packet format employed by 4.2BSD UNIX (among others) in which the data contents of packets sent on the physical network are rearranged. A trailer encapsulation, or "trailer", may be generated by a system under certain conditions in an effort to minimize the number and size of memory-to-memory copy operations performed by a receiving host when processing a data packet. Trailers are strictly a link level packet format and are not visible (when properly implemented) in any higher level protocol processing. This note cites the motivation behind the trailer encapsulation and describes the trailer encapsulation packet formats currently in use on 3 Mb/s Experimental Ethernet, 10 Mb/s Ethernet, and 10 Mb/s V2LNI ring networks.

The use of a trailer encapsulation was suggested by Greg Chesson, and the encapsulation described here was designed by Bill Joy.

RFC-893 Trailer Encapsulations

Motivation

Trailers are motivated by the overhead which may be incurred during protocol processing when one or more memory to memory copies must be performed. Copying can be required at many levels of processing, from moving data between the network medium and the host's memory, to passing data between the operating system and user address spaces. An optimal network implementation would expect to incur zero copy operations between delivery of a data packet into host memory and presentation of the appropriate data to the receiving process. While many packets may not be processed without some copying operations, when the host computer provides suitable memory management support it may often be possible to avoid copying simply by manipulating the appropriate virtual memory hardware.

In a page mapped virtual memory environment, two prerequisites are usually required to achieve the goal of zero copy operations during packet processing. Data destined for a receiving agent must be aligned on a page boundary and must have a size which is a multiple of the hardware page size (or filled to a page boundary). The latter restriction assumes virtual memory protection is maintained at the page level; different architectures may alter these prerequisites.

Data to be transmitted across a network may easily be segmented in the appropriate size, but unless the encapsulating protocol header information is fixed in size, alignment to a page boundary is virtually impossible. Protocol header information may vary in size due to the use of multiple protocols (each with a different header), or it may vary in size by agreement (for example, when optional information is included in the header). To insure page alignment the header information which prefixes data destined for the receiver must be reduced to a fixed size; this is normally the case at the link level of a network. By taking all (possibly) variable length header information and moving it after the data segment a sending host may "do its best" in allowing the receiving host the opportunity to receive data on a page aligned boundary. This rearrangement of data at the link level to force variable length header information to "trail" the data is the substance of the trailer encapsulation.

There are several implicit assumptions in the above argument.

1. The receiving host must be willing to accept trailers.
2. The cost of receiving data on a page aligned boundary should be comparable to receiving data on a non-page aligned boundary. If the overhead of insuring proper alignment is too high, the savings in avoiding copy operations may not be cost effective.
3. The size of the variable length header information should be significantly less than that of the data segment being transmitted. It is possible to move trailing information without physically copying it, but often implementation constraints and the characteristics of the underlying network hardware preclude merely remapping the header(s).
4. The memory to memory copying overhead which is expected to be performed by the receiver must be significant enough to warrant the added complexity in the both the sending and receiving host software.

The first point is addressed by negotiating for the use of trailers at the time of an initial ARP exchange(as defined in RFC-826).

The second point is (to our knowledge) insignificant. While a host may not be

able to take advantage of the alignment and size properties of a trailer packet, it should nonetheless never hamper it.

Regarding the third point, let us assume the trailing header information is copied and not remapped, and consider the header overhead in the TCP/IP protocols as a representative example. If we assume both the TCP and IP protocol headers are part of the variable length header information, then the smallest trailer packet (generated by a VAX) would have 512 bytes of data and 40+ bytes of header information (plus the trailer header described later). While the trailing header could have IP and/or TCP options included this would normally be rare (one would expect most TCP options, for example, to be included in the initial connection setup exchange) and certainly much smaller than 512 bytes. If the data segment is larger, the ratio decreases and the expected gain due to fewer copies on the receiving end increases. Given the relative overheads of a memory to memory copy operation and that of a page map manipulation (including translation buffer invalidation), the advantage is obvious.

The fourth issue, we believe, is actually a non-issue. In our implementation the additional code required to support the trailer encapsulation amounts to about a dozen lines of code in each link level "network interface driver". The resulting performance improvement more than warrants this minor investment in software.

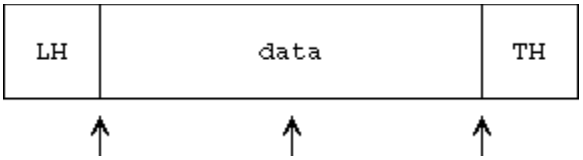
It should be recognized that modifying the network (and normal link) level format of a packet in the manner described forces the receiving host to buffer the entire packet before processing. Clever implementations may parse protocol headers as the packet arrives to find out the actual size (or network level packet type) of an incoming message. This allows these implementations to avoid preallocating maximum sized buffers to incoming packets which it can recognize as unacceptable. Implementations which parses the network level format on the fly are violating layering principles which have been extolled in design for some time (but often violated in implementation). The problem of postponing link level type recognition is a valid criticism. In the case of network hardware which supports DMA, however, the entire packet is always received before processing begins.

RFC-893 Trailer Encapsulations

Trailer Encapsulation Packet Formats

In this section we describe the link level packet formats used on the 3 Mb/s Experimental Ethernet, and 10 Mb/s Ethernet networks as well as the 10 Mb/s V2LNI ring network. The formats used in each case differ only in the format and type field values used in each of the local area network headers.

The format of a trailer packet is shown in the following diagram.



LH:

The fixed-size local network header. For 10 a Mb/s Ethernet, the 16-byte Ethernet header. The type field in the header indicates that both the packet type (trailer) and the length of the data segment.

For the 10 Mb/s Ethernet, the types are between 1001 and 1010 hexadecimal (4096 and 4112 decimal). The type is calculated as 1000 (hex) plus the number of 512-byte pages of data. A maximum of 16 pages of data may be transmitted in a single trailer packet (8192 bytes).

data:

The "data" portion of the packet. This is normally only data to be delivered to the receiving processes (i.e. it contains no TCP or IP header information). Data size is always a multiple of 512 bytes.

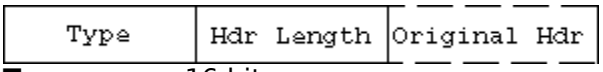
TH:

The "trailer". This is actually a composition of the original protocol headers and a fixed size trailer prefix which defines the type and size of the trailing data. The format of a trailer is shown below.

The arrows indicate the page boundaries on which the receiving host would place its input buffer for optimal alignment when receiving a trailer packet. The link level receiving routine is able to locate the trailer using the size indicated in the link level header's type field. The receiving routine is expected to discard the link level header and trailer prefix, and remap the trailing data segment to the front of the packet to regenerate the original network level packet format.

RFC-893 Trailer Encapsulations

Trailer Format



Type: 16 bits

The type field encodes the original link level type of the transmitted packet. This is the value which would normally be placed in the link level header if a trailer were not generated.

Header length: 16 bits

The header length field of the trailer data segment. This specifies the length in bytes of the following header data.

Original headers: <variable length>

The header information which logically belongs before the data segment. This is normally the network and transport level protocol headers.

Negotiation

On an Ethernet, packets encapsulated with trailers use a distinct Ethernet type, and trailer negotiation is performed at the time that ARP is used to discover the link-layer address of a destination system.

Specifically, the ARP exchange is completed in the usual manner using the normal IP protocol type, but a host that wants to speak trailers will send an additional "trailer ARP reply" packet, i.e., an ARP reply that specifies the trailer encapsulation protocol type but otherwise has the format of a normal ARP reply. If a host configured to use trailers receives a trailer ARP reply message from a remote machine, it can add that machine to the list of machines that understand trailers, e.g., by marking the corresponding entry in the ARP cache.

Hosts wishing to receive trailer encapsulations send trailer ARP replies whenever they complete exchanges of normal ARP messages for IP. Thus, a host that received an ARP request for its IP protocol address would send a trailer ARP reply in addition to the normal IP ARP reply; a host that sent the IP ARP request would send a trailer ARP reply when it received the corresponding IP ARP reply. In this way, either the requesting or responding host in an IP ARP exchange may request that it receive trailer encapsulations.

This scheme, using extra trailer ARP reply packets rather than sending an ARP request for the trailer protocol type, was designed to avoid a continuous exchange of ARP packets with a misbehaving host that, contrary to any specification or common sense, responded to an ARP reply for trailers with another ARP reply for IP. This problem is avoided by sending a trailer ARP reply in response to an IP ARP reply only when the IP ARP reply answers an outstanding request; this is true when the hardware address for the host is still unknown when the IP ARP reply is received. A trailer ARP reply may always be sent along with an IP ARP reply responding to an IP ARP request.

Summary

A link level encapsulation which promotes alignment properties necessary for the efficient use of virtual memory hardware facilities has been described. This encapsulation format is in use on many systems and is a standard facility in 4.2BSD UNIX. The encapsulation provides an efficient mechanism by which cooperating hosts on a local network may obtain significant performance improvements. The use of this encapsulation technique currently requires uniform cooperation from all hosts on a network; hopefully a per host negotiation mechanism may be added to allow consenting hosts to utilize the encapsulation in a non-uniform environment.

RFC-894 A Standard for the Transmission of IP Datagrams over Ethernet Networks

Charles Hornig
Symbolics Cambridge Research Center
April 1984

Status of this Memo

This RFC specifies a standard method of encapsulating Internet Protocol (IP) datagrams on an Ethernet [2]. This RFC specifies a standard protocol for the ARPA-Internet community.

RFC-894 IP Encapsulation for Ethernets

Introduction

This memo applies to the Ethernet (10-megabit/second, 48-bit addresses). The procedure for transmission of IP datagrams on the Experimental Ethernet (3-megabit/second, 8-bit addresses) is described in [3].

RFC-894 IP Encapsulation for Ethernets

Frame Format

IP datagrams are transmitted in standard Ethernet frames. The type field of the Ethernet frame must contain the value hexadecimal 0800. The data field contains the IP header followed immediately by the IP data.

The minimum length of the data field of a packet sent over an Ethernet is 46 octets. If necessary, the data field should be padded (with octets of zero) to meet the Ethernet minimum frame size. This padding is not part of the IP packet and is not included in the total length field of the IP header.

The minimum length of the data field of a packet sent over an Ethernet is 1500 octets, thus the maximum length of an IP datagram sent over an Ethernet is 1500 octets. Implementations are encouraged to support full-length packets. Gateway implementations MUST be prepared to accept full-length packets and fragment them if necessary. If a system cannot receive full-length packets, it should take steps to discourage others from sending them, such as using the TCP Maximum Segment Size option [4].

Note: Datagrams on the Ethernet may be longer than the general Internet default maximum packet size of 576 octets. Hosts connected to an Ethernet should keep this in mind when sending datagrams to hosts not on the same Ethernet. It may be appropriate to send smaller datagrams to avoid unnecessary fragmentation at intermediate gateways. Please see [4] for further information on this point.

Address Mappings

The mapping of 32-bit Internet addresses to 48-bit Ethernet addresses can be done several ways. A static table could be used, or a dynamic discovery procedure could be used.

Static Table

> Each host could be provided with a table of all other hosts on the local network with both their Ethernet and Internet addresses.

Dynamic Discovery

> Mappings between 32-bit Internet addresses and 48-bit Ethernet addresses could be accomplished through the Address Resolution Protocol (ARP). Internet addresses are assigned arbitrarily on some Internet network. Each host's implementation must know its own Internet address and respond to Ethernet Address Resolution packets appropriately. It should also use ARP to translate Internet addresses to Ethernet addresses when needed.

Broadcast Address

> The broadcast Internet address (the address on that network with a host part of all binary ones) should be mapped to the broadcast Ethernet address (of all binary ones, FF-FF-FF-FF-FF-FF hex).

The use of the ARP dynamic discovery procedure is strongly recommended.

Trailer Formats

Some versions of Unix 4.2bsd use a different encapsulation method in order to get better network performance with the VAX virtual memory architecture. Consenting systems on the same Ethernet may use this format between themselves.

No host is required to implement it, and no datagrams in this format should be sent to any host unless the sender has positive knowledge that the recipient will be able to interpret them, such knowledge being gained by negotiation. Details of the **Trailer Encapsulation** may be found in **RFC-893**.

Byte Order

As described in Appendix B of the Internet Protocol specification, the IP datagram is transmitted over the Ethernet as a series of 8-bit bytes.

References

[1] Postel, J., "Internet Protocol", RFC-791, USC/Information Sciences Institute, September 1981.

[2] "The Ethernet - A Local Area Network", Version 1.0, Digital Equipment Corporation, Intel Corporation, Xerox Corporation, September 1980.

[3] Postel, J., "A Standard for the Transmission of IP Datagrams over Experimental Ethernet Networks", RFC-895, USC/Information Sciences Institute, April 1984.

[4] Postel, J., "The TCP Maximum Segment Size Option and Related Topics", RFC-879, USC/Information Sciences Institute, November 1983.

[5] Plummer, D., "An Ethernet Address Resolution Protocol", RFC-826, Symbolics Cambridge Research Center, November 1982.

[6] Leffler, S., and M. Karels, "Trailer Encapsulations", RFC-893, University of California at Berkeley, April 1984.

Reverse Address Resolution Protocol (RARP)

Introduction
Network hosts such as diskless workstations frequently do not know their protocol addresses when booted; they often know only their hardware interface addresses. To communicate using higher-level protocols like IP, they must discover their protocol address from some external source. Our problem is that there is no standard mechanism for doing so.

Plummer's "Address Resolution Protocol" (ARP) [RFC-826] is designed to solve a complementary problem, resolving a host's hardware address given its protocol address. As with ARP, we assume a broadcast medium, such as Ethernet.

Design Considerations

The following considerations guided our design of the RARP protocol.

A ARP and RARP are different operations. ARP assumes that every host knows the mapping between its own hardware address and protocol address(es). Information gathered about other hosts is accumulated in a small cache. All hosts are equal in status; there is no distinction between clients and servers.

On the other hand, RARP requires one or more server hosts to maintain a database of mappings from hardware address to protocol address and respond to requests from client hosts.

B As mentioned, RARP requires that server hosts maintain large databases. It is undesirable and in some cases impossible to maintain such a database in the kernel of a host's operating system. Thus, most implementations will require some form of interaction with a program outside the kernel.

C Ease of implementation and minimal impact on existing host software are important. It would be a mistake to design a protocol that required modifications to every host's software, whether or not it intended to participate.

D It is desirable to allow for the possibility of sharing code with existing software, to minimize overhead and development costs.

RFC-903 Reverse Address Resolution Protocol

Protocol Description

RARP is specified as a separate protocol at the data-link level. For example, if the medium used is Ethernet, then RARP packets will have an Ethertype (8035H) different from that of ARP(806H). This recognizes that ARP and RARP are two fundamentally different operations, not supported equally by all hosts. The impact on existing systems is minimized; existing ARP servers will not be confused by RARP packets. It makes RARP a general facility that can be used for mapping hardware addresses to any higher level protocol address.

This approach provides the simplest implementation for RARP client hosts, but also provides the most difficulties for RARP server hosts. However, these difficulties should not be insurmountable, as is shown in Examples, where we sketch two possible implementations for 4.2BSD Unix.

RARP uses the same packet format that is used by ARP, namely:

ar$hrd (hardware address space) - 16 bits

ar$pro (protocol address space) - 16 bits

ar$hln (hardware address length) - 8 bits

ar$pln (protocol address length) - 8 bits

ar$op (opcode) - 16 bits

ar$sha (source hardware address) - n bytes,

where n is from the ar$hln field.

ar$spa (source protocol address) - m bytes,

where m is from the ar$pln field.

ar$tha (target hardware address) - n bytes

ar$tpa (target protocol address) - m bytes

arhrd, arpro, ar$hln and ar$pln are the same as in regular ARP.

See Discussion.

Protocol Description
Discussion

Suppose, for example, that 'hardware' addresses are 48-bit Ethernet addresses, and 'protocol' addresses are 32-bit Internet Addresses. That is, we wish to determine Internet Addresses corresponding to known Ethernet addresses. Then, in each RARP packet, ar$hrd = 1 (Ethernet), ar$pro = 2048 decimal (the Ethertype of IP packets), ar$hln = 6, and ar$pln = 4.

There are two opcodes: 3 ('request reverse') and 4 ('reply reverse'). An opcode of 1 or 2 has the same meaning as in ARP; packets with such opcodes may be passed on to regular ARP code. A packet with any other opcode is undefined. As in ARP, there are no "not found" or "error" packets, since many RARP servers are free to respond to a request. The sender of a RARP request packet should timeout if it does not receive a reply for this request within a reasonable amount of time.

The arsha, arspa, artha, and ar$tpa fields of the RARP packet are interpreted as follows:

When the opcode is 3 ('request reverse'):

ar$sha is the hardware address of the sender of the packet.

ar$spa is undefined.

ar$tha is the 'target' hardware address.

In the case where the sender wishes to determine his own protocol address, this, like ar$sha, will be the hardware address of the sender.

ar$tpa is undefined.

When the opcode is 4 ('reply reverse'):

ar$sha is the hardware address of the responder (the sender of the reply packet).

ar$spa is the protocol address of the responder (see the note below).

ar$tha is the hardware address of the target, and should be the same as that which was given in the request.

ar$tpa is the protocol address of the target, that is, the desired address.

Note that the requirement that ar$spa in opcode 4 packets be filled in with the responder's protocol is purely for convenience. For instance, if a system were to use both ARP and RARP, then the inclusion of the valid protocol-hardware address pair (arspa, arsha) may eliminate the need for a subsequent ARP request.

Two Example Implementations of RARP for 4.2BSD Unix

The following implementation sketches outline two different approaches to implementing a RARP server under 4.2BSD.

A Provide access to data-link level packets outside the kernel. The RARP server is implemented completely outside the kernel and interacts with the kernel only to receive and send RARP packets. The kernel has to be modified to provide the appropriate access for these packets; currently the 4.2 kernel allows access only to IP packets. One existing mechanism that provides this capability is the CMU "packet-filter" pseudo driver. This has been used successfully at CMU and Stanford to implement similar sorts of "user-level" network servers.

B Maintain a cache of database entries inside the kernel. The full RARP server database is maintained outside the kernel by a user process. The RARP server itself is implemented directly in the kernel and employs a small cache of database entries for its responses. This cache could be the same as is used for forward ARP.
The cache gets filled from the actual RARP database by means of two new ioctls. (These are like SIOCIFADDR, in that they are not really associated with a specific socket.) One means: "sleep until there is a translation to be done, then pass the request out to the user process"; the other means: "enter this translation into the kernel table". Thus, when the kernel can't find an entry in the cache, it puts the request on a (global) queue and then does a wakeup(). The implementation of the first ioctl is to sleep() and then pull the first item off of this queue and return it to the user process. Since the kernel can't wait around at interrupt level until the user process replies, it can either give up (and assume that the requesting host will retransmit the request packet after a second) or if the second ioctl passes a copy of the request back into the kernel, formulate and send a response at that time.

RFC-904
Exterior Gateway Protocol Formal Specification

D.L. Mills

April 1984

This RFC is the specification of the Exterior Gateway Protocol (EGP). This document updates RFCs 827 and 888. This RFC specifies a standard for the DARPA community. Interactions between gateways of different autonomous systems in the ARPA-Internet must follow this protocol.

RFC-904 Exterior Gateway Protocol Formal Specification

Introduction

This document is a formal specification of the Exterior Gateway Protocol (EGP), which is used to exchange net-reachability information between Internet gateways belonging to the same or different autonomous systems. The specification is intended as a reference guide for implementation, testing and verification and includes suggested algorithmic parameters suitable for operation over a wide set of configurations, including the ARPANET and many local-network technologies now part of the Internet system.

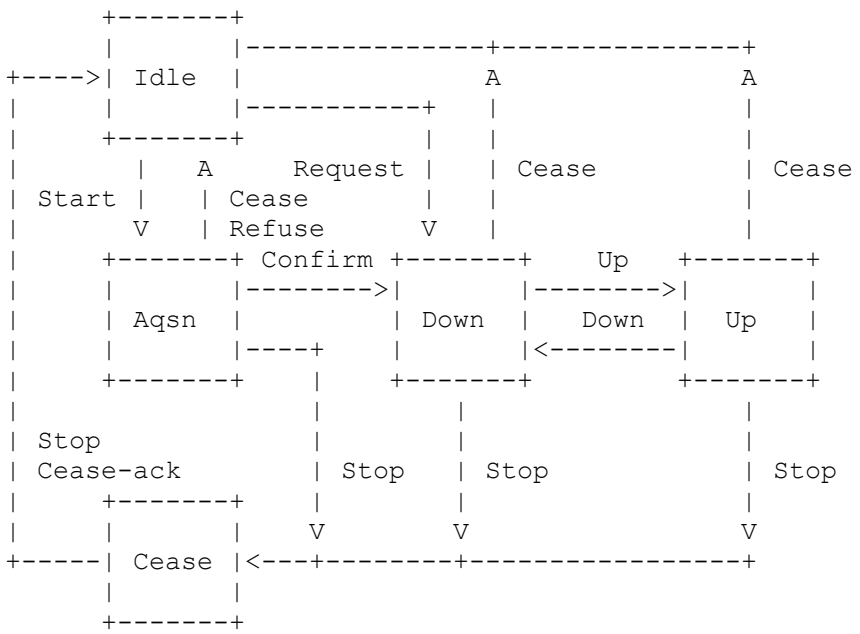
Specifically excluded in this document is discussion on the background, application and limitations of EGP, which have been discussed elsewhere (RFC-827, RFC-888). If, as expected, EGP evolves to include topologies not restricted to tree-structures and to incorporate full routing capabilities, this specification will be amended or obsoleted accordingly. However, it is expected that, as new features are added to EGP, the basic protocol mechanisms described here will remain substantially unchanged, with only the format and interpretation of the Update message (see below) changed.

Section 2 of this document describes the nomenclature used, while Section 3 describes the state-machine model, including events, actions, parameters and state transitions. Section 4 contains a functional description of the operation of the machine, together with specific procedures and algorithms. Appendix A describes the EGP message formats, while Appendix B contains a summary of the minor differences between these and the formats described in RFC-888. Appendix C presents a reachability analysis including a table of composite state transitions for a system of two communicating EGP gateways.

Summary and Overview

EGP exists in order to convey net-reachability information between neighboring gateways, possibly in different autonomous systems. The protocol includes mechanisms to acquire neighbors, monitor neighbor reachability and exchange net-reachability information in the form of Update messages. The protocol is based on periodic polling using Hello/I-Heard-You (I-H-U) message exchanges to monitor neighbor reachability and Poll commands to solicit Update responses.

Specification of EGP is based on a formal model consisting of a finite-state automaton with defined events, state transitions and actions. The following diagram shows a simplified graphical representation of this machine (see Section 3.4 for a detailed state transition table).



Following is a brief summary and overview of gateway operations by state as determined by this model.

Idle State (0)

In the Idle state the gateway has no resources (table space) assigned to the neighbor and no protocol activity of any kind is in progress. It responds only to a Request command or a Start event (system or operator initiated) and ignores all other commands and responses. The gateway may optionally return a Cease-ack response to a Cease command in this state.

Upon receipt of a Request command the gateway initializes the state variables as described in the State Variables Section, sends a Confirm response and transitions to the Down state, if resource commitments permit, or sends a Refuse response and returns to the Idle state if not. Upon receipt of a Start event it sends a Request command and transitions to the Acquisition state.

Acquisition State (1)

In the Acquisition state the gateway periodically retransmits Request commands. Upon receiving a Confirm response it initializes the state variables and transitions to the Down state. Upon receiving a

Refuse response it returns to the Idle state. The gateway does not send any other commands or responses in this state, since not all state variables have yet been initialized.

Down State (2)

In the Down state the gateway has received a Request command or a Confirm response has been received for a previously sent Request. The neighbor-reachability protocol has declared the neighbor to be down. In this state the gateway processes Request, Cease and Hello commands and responds as required. It periodically retransmits Hello commands if enabled. It does not process Poll commands and does not send them, but may optionally process an unsolicited Update indication.

Up State (3)

In the Up state the neighbor-reachability protocol has declared the neighbor to be up. In this state the gateway processes and responds to all commands. It periodically retransmits Hello commands, if enabled, and Poll commands.

Cease State (4)

A Stop event causes a Cease command to be sent and a transition to the Cease state. In this state the gateway periodically retransmits the Cease command and returns to the Idle state upon receiving a Cease-ack response or a another Stop event. The defined state transitions are designed to ensure that the neighbor does with high probability receive the Cease command and stop the protocol.

In following sections of this document a state machine which can serve as a model for implementation is described. It may happen that implementators may deviate from this model while conforming to the protocol specification; however, in order to verify conformance to the specification, the state-machine model is intended as the reference model.

Although not mentioned specifically in this document, it should be understood that all Internet gateways must include support for the Internet Control Message Protocol (ICMP), specifically ICMP Redirect and ICMP Destination Unreachable messages.

Nomenclature

The following EGP message types are recognized in this document. The format of each of these messages is described in Appendix A.

Name	Function
Request	request acquisition of neighbor and/or initialize polling variables
Confirm	confirm acquisition of neighbor and/or initialize polling variables
Refuse	refuse acquisition of neighbor
Cease	request de-acquisition of neighbor
Cease-ack	confirm de-acquisition of neighbor
Hello	request neighbor reachability
I-H-U	confirm neighbor reachability
Poll	request net-reachability update
Update	net-reachability update
Error	error

EGP messages are classed as commands which request some action, responses, which are sent to indicate the status of that action, and indications, which are similar to responses, but may be sent at any time. Following is a list of commands along with their possible responses.

Command	Corresponding Responses
Request	Confirm, Refuse, Error
Cease	Cease-ack, Error
Hello	I-H-U, Error
Poll	Update, Error

The Update and Error messages are classed both as responses and indications. When sent in reply to a previous command, either of these messages is classed as a response. In some circumstances an unsolicited Update message can be sent, in which case it is classed as an indication. The use of the Error message other than as a response to a previous command is a topic for further study.

State Machine

This section describes the state-machine model for EGP, including the variables and constants which establish the state at any time, the events which cause the state transitions, the actions which result from these transitions and the state-transition table which defines the behavior.

State Variables

The state-machine model includes a number of state variables which establish the state of the protocol between the gateway and each of its neighbors. Thus, a gateway maintaining EGP with a number of neighbors must maintain a separate set of these state variables for each neighbor. The current state, events and actions of the state machine apply to each neighbor separately.

The model assumes that system resources, including the set of state variables, are allocated when the state machine leaves the Idle state, either because of the arrival of a Request specifying a new neighbor addreess, or because of a Start event specifying a new neighbor address. When either of these events occur the values of the state variables are initialized as indicated below. Upon return to the Idle state all resources, including the set of state variables, are deallocated and returned to the system. Implementators may, of course, elect to dedicate resources and state variables permananently.

Included among the set of state variables are the following which determine the state transitions of the model. Initial values for all of the variables except the send sequence number S are set during the initial Request/Confirm exchange. The initial value for S is arbitrary.

Name	**Function**
R	receive sequence number
S	send sequence number
T1	interval between Hello command retransmissions
T2	interval between Poll command retransmissions
T3	interval during which neighbor-reachability indications are counted
M	hello polling mode
t1	timer 1 (used to control Request, Hello and Cease command retransmissions)
t2	timer 2 (used to control Poll command retransmissions)
t3	timer 3 (abort timer)

Additional state variables may be necessary to support various timer and similar internal housekeeping functions. The function and management of the cited variables are discussed in Section 4.

Fixed Parameters

This section defines several fixed parameters which characterize the gateway functions. Included is a suggested value for each parameter based on experimental implementations in the Internet system. These values may or may not be appropriate for the individual configuration.

Following is a list of time-interval parameters which control retransmissions and other time-dependent functions.

Name	Value	Description
P1	30 sec	minimum interval acceptable between successive Hello commands received
P2	2 min	minimum interval acceptable between successive Poll commands recieved
P3	30 sec	interval between Request or Cease command retransmissions
P4	1 hr	interval during which state variables are maintained in the absence of commands or responses in the Down and Up states.
P5	2 min	interval during which state variables are maintained in the absence of responses in the Acquisition and Cease states

Parameters P4 and P5 are used only if the abort-timer option is implemented. Parameter P4 establishes how long the machine will remain in the Down and Up states in the absence of commands or responses and would ordinarily be set to sustain state information while the neighbor is dumped and restarted, for example. Parameter P5 establishes how long the machine will remain in the Acquisition or Cease states in the absence of responses and would ordinarily be set in the same order as the expected value of T3 variables.

Following is a list of other parameters of interest.

Name	Active	Passive	Description
j	3	1	neighbor-up threshold
k	1	4	neighbor-down threshold

The j and k parameters establish the "noise immunity" of the neighbor-reachability protocol described later. The values in the Active column are suggested if the gateway elects to do hello polling, while the values in the Passive column are suggested otherwise.

Events

Following is a list of events that can cause state transitions in the model.

Name	Event
Up	At least j neighbor-reachability indications have been received within the last T3 seconds.
Down	At most k neighbor-reachabilitiy indications have been received within the last T3 seconds.
Request	Request command has been received.
Confirm	Confirm command has been received.
Refuse	Refuse response has been received.
Cease	Cease command has been received.
Cease-ack	Cease-ack response has been received.
Hello	Hello command has been received.
I-H-U	I-H-U response has been received.
Poll	Poll command has been received.
Update	Update response has been received.
Start	Start event has been recognized due to system or operator intervention.
Stop/t3	Stop event has been recognized due to (a) system or operator intervention or (b) expiration of the abort timer t3.
t1	Timer t1 has counted down to zero.
t2	Timer t2 has counted down to zero.

There is one special event, called a neighbor-reachability indication, which occurs when:

- The gateway is operating in the active mode (hello polling enabled) and either a Confirm, I-H-U or Update response is received.
- The gateway is operating in the passive mode (hello polling disabled) and either a Hello or Poll command is received with the "Up state" code in the Status field.

State Transition Table

The following table summarizes the state transitions that can occur in response to the events listed above. Transitions are shown in the form n/a, where n is the next state and a represents the action.

	0 Idle	1 Aqsn	2 Down	3 Up	4 Cease
Up	0	1	3/Poll	3	4
Down	0	1	2	2	4
Request	2/Confirm *	2/Confirm	2/Confirm	2/Confirm	4/Cease
Confirm	0/Cease **	2	2	3	4
Refuse	0/Cease **	0	2	3	4
Cease	0/Cease-ack	0/Cease-ack	0/Cease-ack	0/Cease-ack	0/Cease-ack
Cease-ack	0	1	2	3	0
Hello	0/Cease **	1	2/I-H-U	3/I-H-U	4
I-H-U	0/Cease **	1	2/Process	3/Process	4
Poll	0/Cease **	1	2	3/Update	4
Update	0/Cease **	1	2	3/Process	4
Start	1/Request	1/Request	1/Request	1/Request	4
Stop/t3	0	0	4/Cease	4/Cease	0
t1	0	1/Request	2/Hello	3/Hello	4/Cease
t2	0	1	2	3/Poll	4

Note *: The transition shown applies to the case where the neighbor-acquisition request is accepted. The transition "0/Refuse" applies to the case where the request is rejected.

Note **: The Cease action shown is optional.

State Transitions and Actions

The following table describes in detail the transitions of the state machine and the actions evoked.

Event	Next State	Message Sent	Actions
Idle State (0)			
Request	2	Confirm Hello	Initialize state variables and reset timer t1 to T1 seconds and reset timer t3 to P5 seconds.
(or)	0	Refuse	Return resources.
Cease	0	Cease-ack	Return resources.
Start	1	Request	Reset timer t1 to P3 seconds and reset timer t3 to P5 seconds.
Acquisition State (1)			
Request	2	Confirm Hello	Initialize state variables and reset timer t1 to T1 seconds and reset timer t3 to P5 seconds.
Confirm	2	Hello	Initialize state variables and reset timer t1 to T1 seconds and reset timer t3 to P5 seconds.
Refuse	0		Stop timers and return resources.
Cease	0	Cease-ack	Stop timers and return resources.
Start	1	Request	Reset timer t1 to P3 seconds and reset timer t3 to P5 seconds.
Stop/t3	0		Stop timers and return resources.
t1	1	Request	Reset timer t1 to P3 seconds.
Down State (2)			
Up	3	Poll	Reset timer t2 to T2 seconds.
Request	2	Confirm Hello	Reinitialize state variables and reset timer t1 to T1 seconds and reset timer t3 to P5 seconds.
Cease	0	Cease-ack	Stop timers and return resources.
Hello	2	I-H-U	
I-H-U	2		Process neighbor-reachability info.
Start	1	Request	Reset timer t1 to P3 seconds and reset timer t3 to P5 seconds.
Stop/t3	4	Cease	Reset timer t1 to P3 seconds and reset timer t3 to P5 seconds.
t1	2	Hello	Reset timer t1 to T1 seconds.
Up State (3)			

Note: Reset timer t3 to P4 seconds on receipt of a reachability indication.

Note: Reset timer t3 to P4 seconds on receipt of a reachability indication.

Down	2		Stop timer t2.
Request	2	Confirm Hello	Renitialize state variables and reset timer t1 to T1 seconds and reset timer t3 to P5 seconds.
Cease	0	Cease-ack	Stop timers and return resources.
Hello	3	I-H-U	
I-H-U	3		Process neighbor-reachability info.
Poll	3	Update	
Update	3		Process net-reachability info.
Start	1	Request	Reset timer t1 to P3 seconds and reset timer t3 to P5 seconds.
Stop/t3	4	Cease	Reset timer t1 to P3 seconds and reset timer t3 to P5 seconds.
t1	3	Hello	Reset timer t1 to T1 seconds.
t2	3	Poll	Reset timer t2 to T2 seconds.

Cease State (4)

Request	4	Cease	
Cease	0	Cease-ack	Stop timers and return resources.
Cease-ack	0		Stop timers and return resources.
Stop/t3	0		Stop timers and return resources.
t1	4	Cease	Reset timer t1 to P3 seconds.

Functional Description

This section contains detailed descriptions of the various procedures and algorithms used to manage the protocol.

Managing the State Variables

The state variables which characterize the protocol are summarized above. This section describes the detailed management of these variables, including sequence numbers, polling intervals and timers.

Sequence Numbers

All EGP commands and replies carry a sequence number. The state variable R records the last sequence number received in a command from that neighbor. The current value of R is used as the sequence number for all replies and indications sent to the neighbor until a command with a different sequence number is received from that neighbor.

Implementors are free to manage the sequence numbers of the commands sent; however, it is suggested that a separate send state variable S be maintained for each EGP neighbor and that its value be incremented just before the time an Poll command is sent and at no other times. The actions upon receipt of a response or indication with sequence number not equal to S is not specified; however, it is recommended these be discarded.

Polling Intervals

As part of the Request/Confirm exchange a set of polling intervals are established including T1, which establishes the interval between Hello command retransmissions, and T2, which establishes the interval between Poll retransmissions.

Each gateway configuration is characterized by a set of fixed parameters, including P1, which specifies the minimum polling interval at which it will respond to Hello commands, and P2, which specifies the minimum polling interval at which it will respond to Poll commands. P1 and P2 are inserted in the Hello Interval (S1) and Poll Interval (S2) fields, respectively, of Request commands and Confirm responses.

A gateway receiving a Request command or Confirm response uses the S1 and S2 fields in the message to calculate its own T1 and T2 state variables, respectively. Implementors are free to perform this calculation in arbitrary ways; however, the following constraints must be observed:

1) If T1 < S1 the neighbor may discard Hello commands. If T2 < S2 the neighbor may discard Poll commands.

2) The time window T3 in which neighbor-reachability indications are counted is dependent on T1. In the case where two neighbors select widely differing values for their T3 state variables, the neighbor-reachability algorithm may not work properly. This can be avoided if T1 > max(P1, S1).

3) If either S1 or S2 or both are unacceptable for some reason (e.g. exceed useful limits), the neighbor may either send a Refuse response or declare a Stop event, depending on state.

It is suggested that T3 be computed as four times the value of T1, giving a window of four neighbor-reachability indications, which has been found appropriate in the experimental implementations. Implementors may choose to make T3 a fixed parameter in those cases where the path between the neighbors has well-known characteristics.

Note that, if a gateway attempts to send Hello commands near the rate max(P1, S1) or Poll commands near the rate max(P2, S2), the neighbor may observe their succeeding arrivals to violate the polling restrictions due to bunching in the net. For this reason the gateway should send at rates somewhat below these. Just how much below these rates is appropriate depends on many factors beyond the scope of this specification.

Hello Polling Mode

The neighbor-reachability algorithm can be used in either the active or passive mode. In the active mode Hello commands are sent periodically along with Poll commands, with reachability determined by the corresponding I-H-U and Update responses. In the passive mode Hello commands are not sent and I-H-U responses are not expected. Reachability is then determined from the Status field of received Hello or Poll commands or Update responses.

The M state variable specifies whether the gateway operates in the active or passive mode. At least one of the two neighbors sharing the protocol must operate in the active mode; however, the neighbor-reachability protocol is designed to work even if both neighbors operate in the active mode. The value of M is determined from the Status field of a Request command or Confirm response. The sender sets this field according to whether the implementation supports the active mode, passive mode or both:

Status	**Sender capabilities**
0	either active or passive
1	active only
2	passive only

The receiver inspects this field and sets the value of M according to its own capabilities as follows:

Status	**Receiver**	**capabilities**	
field	**0**	**1**	**2**
0	*	active	passive
1	passive	active	passive
2	active	active	**

In the case of "*" the mode is determined by comparing the autonomous system numbers of the neighbors. The neighbor with the smallest such number assumes active mode, while the other neighbor assumes passive mode. In the case of "**" the neighbor may either send a Refuse response or declare a Stop event, depending on state.

Timers

There are three timers defined in the state machine: t1, used to control retransmission of Request, Hello and Cease messages, t2, used to control retransmission of Poll commands, and t3, which serves as an abort-timer mechanism should the protocol hang indefinitely. The timers are set to specified values upon entry to each state and count down to zero.

In the case of t1 and t2 state-dependent events are declared when the timer counts down to zero, after which the timer is reset to the specified value and counts down again. In the case of t3 a Stop event is declared when the timer counts down to zero. Implementors may choose not to implement t3 or, if so, may choose to implement it only in certain states, with the effect that Request, Hello and/or Cease commands may be retransmitted indefinitely.

The following table shows the initial values for each of the timers in each state. A missing value indicates the timer is not used in that state. Note that timer t3 is set to P4 upon receipt of a neighbor-reachability indication when in either the Down or Up states.

	Idle	**Aqsn**	**Down**	**Up**	**Cease**
Timer	**0**	**1**	**2**	**3**	**4**
t1		P3	T1		P3
t2					T2
t3		P5	P5		P5

Starting and Stopping the Protocol

The Start and Stop events are intrinsic to the system environment of the gateway. They can be declared as the result of the gateway process being started and stopped by the operator, for example. A Start event has meaning only in some states; however, a Stop event has meaning in all states.

In all except the Idle state the abort timer t3 is presumed running. This timer is initialized at P5 seconds upon entry to any state and at P4 seconds upon receipt of a neighbor-reachability indication in the Down and Up states. If it expires a Stop event is declared. A Stop event can also be declared by an intrinsic system action such as a resource problem or operator command.

If the abort timer is not implemented a manually-initiated Stop event can be used to stop the protocol. If this is done in the Down or Up states, the machine will transition to the Cease state and emit a Cease command. If the neighbor does not respond to this command the machine will stay in the Cease state indefinitely; however, a second Stop event can be used in this state to force a transition to the Idle state.

A Cease command received in any state will cause the gateway to immediately send the Cease-ack response and transition to the Idle state. This causes the protocol to be stopped and all system resources committed to the gateway process to be released. The interval between the time the gateway enters the Idle state as the result of receiving a Cease command and the time when it next sends a Request command to resume the protocol is not specified; however, it is recommended this interval be at least P5 seconds.

It may happen that the Cease-ack response is lost in the network, causing the neighbor to retransmit the Cease response indefinitely, at least if it has not implemented the abort-timer option. In order to reduce the likelihood of this happening, it is suggested that a gateway in the Idle state be prepared to reply to a Cease command with a Cease-ack response whenever possible.

Determining Neighbor Reachability

The purpose of the neighbor-reachability algorithm is to confirm that the neighbor can safely be considered operational and capable of providing reliable net-reachability information. An equally important purpose is to filter noisy reachability information before sending it on to the remainder of the Internet gateway system, thus avoiding unnecessary reachability changes.

As described above, a gateway operating in the active mode sends periodic Hello commands and listens for I-H-U responses in order to determine neighbor-reachability indications. A gateway operating in the passive mode determines reachability indications by means of the Status field in received Hello commands. Poll commands and Update responses can be used in lieu of Hello commands and I-H-U responses respectively, since they contain the same Status-field information.

The neighbor-reachability algorithm runs continuously while the gateway is in the Down and Up states and operates as follows. Define a moving window in time starting at the present and extending backwards for t seconds. Then count the number n of neighbor-reachability indications which have occurred in that window. If n increases to j, then declare a Up event. If n decreases to k, then declare a Down event. The number n is set to zero upon entering the Down state from any state other than the Up state.

The window t in this algorithm is defined as T3 seconds, the value of which is suggested as four times T1, which itself is determined during the Request/Confirm exchange. For proper operation of the algorithm only one neighbor-reachability indication is significant in any window of T1 seconds and additional ones are ignored. Note that the only way n can increase is as the result of a new neighbor-reachability indication and the only way it can decrease is as the result of an old neighbor-reachability indication moving out of the window.

The behavior of the algorithm described above and using the suggested fixed parameters j and k differs depending on whether the gateway is operating in the active or passive mode. In the active mode (j = 3, k = 1 and T3/T1 = 4), once the neighbor has been declared down it will be forced down for at least two T1 intervals and, once it has been declared up it will be forced up for at least two T1 intervals. It will not change state unless at least three of the last four determinations of reachability have indicated that change.

In the passive mode (j = 1, k = 4 and T3/T1 = 4), the neighbor will be considered up from the first time the Status field of a Hello or Poll command or Update response indicates "Up state" until four successive T1 intervals have passed without such indication. This design, suggested by similar designs used in the ARPANET, has proven effective in the experimental implementations, but may need to be adjusted for other configurations.

It is convenient for the active gateway to send Hello commands at a rate of one every T1 seconds and substitute a Poll command for a Hello command approximately once every T2 seconds, with the neighbor-reachability indication generated by the corresponding I-H-U or Update responses. Its passive neighbor generates neighbor-reachability indications from the Status field of received Hello and Poll commands and Update responses.

Implementors may find the following model useful in the understanding and implementation of this algorithm. Consider an n-bit shift register that shifts one bit to the right each T1-second interval. If a neighbor-reachability indication was received during the preceding T1-second interval a one bit is shifted into the register at the end of the interval; otherwise, a zero bit is shifted. A table of 2**n entries indexed by the contents of the register can be used to calculate the number of one bits, which can then be used to declare the appropriate event to the state machine. A value of n equal to four has been found useful in the experimental implementations.

Determining Network Reachability

Network reachability information is encoded into Update messages in the form of lists of nets and gateways. The IP Source Address field of the Poll command is used to specify a network common to the autonomous systems of each of the neighbors, which is usually, but not necessarily, the one common to the neighbors themselves. The Update response includes a list of gateways on the common net. Associated with each gateway is a list of the networks reachable via that gateway together with corresponding hop counts.

It is important to understand that, at the present state of development as described in RFC-827 and RFC-888, the EGP architectural model restricts the interpretation of "reachable" in this context. This consideration, as well as the implied topological restrictions, are beyond the scope of discussion here. The reader is referred to the RFCs for further discussion.

Two types of gateway lists can be included in the Update response, the format of which is described in Appendix A. Both lists include only those gateways directly connected to the net specified in the IP Source Network field of the last-received Poll command. The internal list includes some or all of the gateways in the same autonomous system as the sender, together with the nets which are reachable via these gateways, with the sending gateway listed first. A net is reachable in this context if a path exists to that net including only gateways in the system. The external list includes those gateways in other autonomous systems known to the sender. It is important to realize that the hop counts do not represent a routing metric and are comparable between different gateways only if those gateways belong to the same autonomous system; that is, are in the internal list.

According to the current system architectural model, only gateways belonging to a designated system, called the core system, may include the external list in their Update responses. All other gateways may include only those gateways belonging to the same system and can claim reachability for a particular net only if that net is reachable in the same system.

The interval between successive Poll commands T2 is determined during the Request/Confirm exchange. However, the specification permits at most one unsolicited Update indication between succeeding Poll commands received from the neighbor. It is the intent of the model here that an Update indication is sent (a) upon entry to the Up state and (b) when a change in the reachability data base is detected, subject to this limitation.

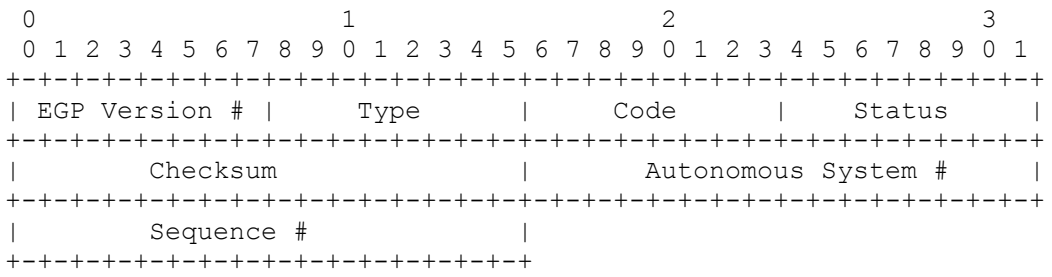
Occasionally it may happen that a Poll command or Update response is lost in the network, with the effect that net-reachability information may not be available until after another T2 interval. As an implementation option, the gateway sending a Poll command and not receiving an Update response after T1 seconds may send another Poll. The gateway receiving this Poll may either (a) send an Update response if it never received the original Poll for that interval, (b) send a second Update response (which counts as the unsolicited Update indication mentioned in the preceding paragraph) or (c) send an Error response or not respond at all in other cases.

Error Messages

Error messages can be used to report problems such as described in Appendix A in connection with the Error Response/Indication message format. In general, an Error message is sent upon receipt of another command or response with bad format, content or ordering, but never in response to another Error message. Receipt of an Error message should be considered advisory and not result in change of state, except possibly to evoke a Stop event.

Appendix A. EGP Message Formats

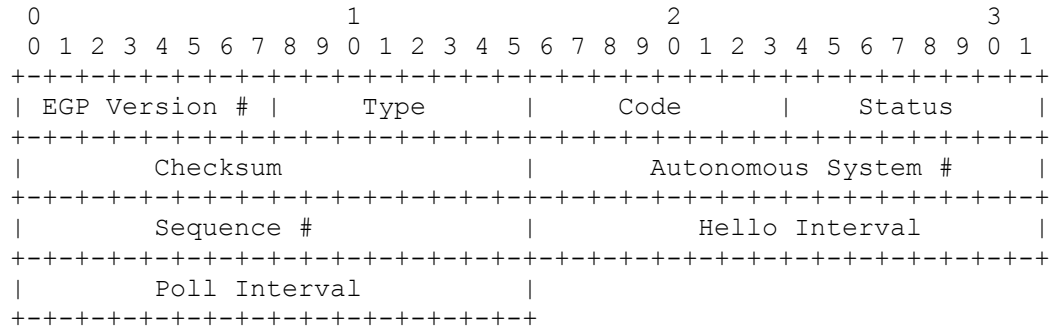
The formats for the various EGP messages are described in this section. All EGP messages include a ten-octet header of six fields, which may be followed by additional fields depending on message type. The format of the header is shown below along with a description of its fields.



EGP Version #	assigned number identifying the EGP version (currently 2)
Type	identifies the message type
Code	identifies the message code (subtype)
Status	contains message-dependent status information
Checksum	The EGP checksum is the 16-bit one's complement of the one's complement sum of the EGP message starting with the EGP version number field. When computing the checksum the checksum field itself should be zero.
Autonomous System #	assigned number identifying the particular autonomous system
Sequence #	send state variable (commands) or receive state variable (responses and indications)

Following sections describe each of the message formats. Note that the above description applies to all formats and will not be repeated.

Neighbor Acquisition Messages



Note: the Hello Interval and Poll Interval fields are present only in Request and Confirm messages.

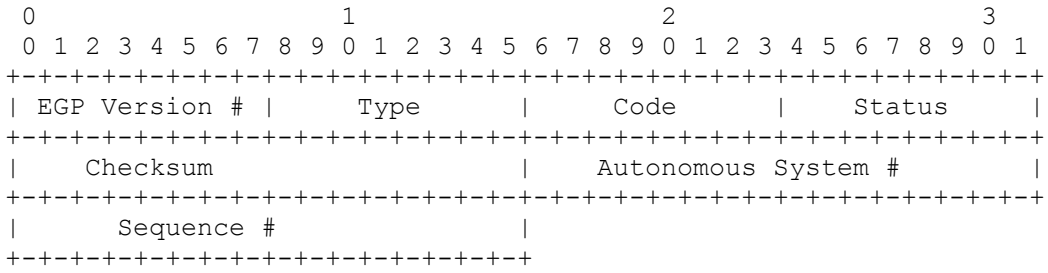
Type	3	
Code	0	Request command
	1	Confirm response
	2	Refuse response
	3	Cease command
	4	Cease-ack response
Status (see below)	0	unspecified
	1	active mode
	2	passive mode
	3	insufficient resources
	4	administratively prohibited
	5	going down
	6	parameter problem
	7	protocol violation
Hello Interval	minimum Hello command polling interval (seconds)	
Poll Interval	minimum Poll command polling interval (seconds)	

Following is a summary of the assigned Status codes along with a list of scenarios in which they might be used.

Code	**Status**	**Scenarios**
0	unspecified	when nothing else fits
1	active mode	Request/Confirm only
2	passive mode	Request/Confirm only
3	insufficient resources	1. out of table space 2. out of system resources
4	administratively prohibited	1. unknown Autonomous System 2. use another gateway

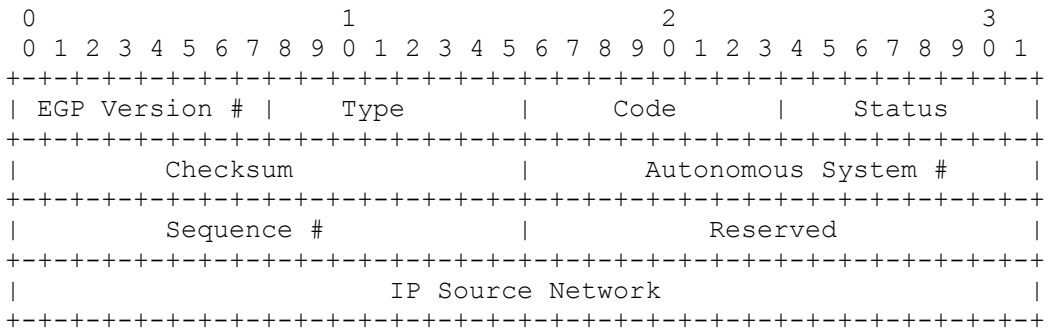
5	going down	1. operator initiated Stop 2. abort timeout
6	parameter problem	1. nonsense polling parameters 2. unable to assume compatible mode
7	protocol violation	1. Invalid command or response received in this state

Neighbor Reachability Messages



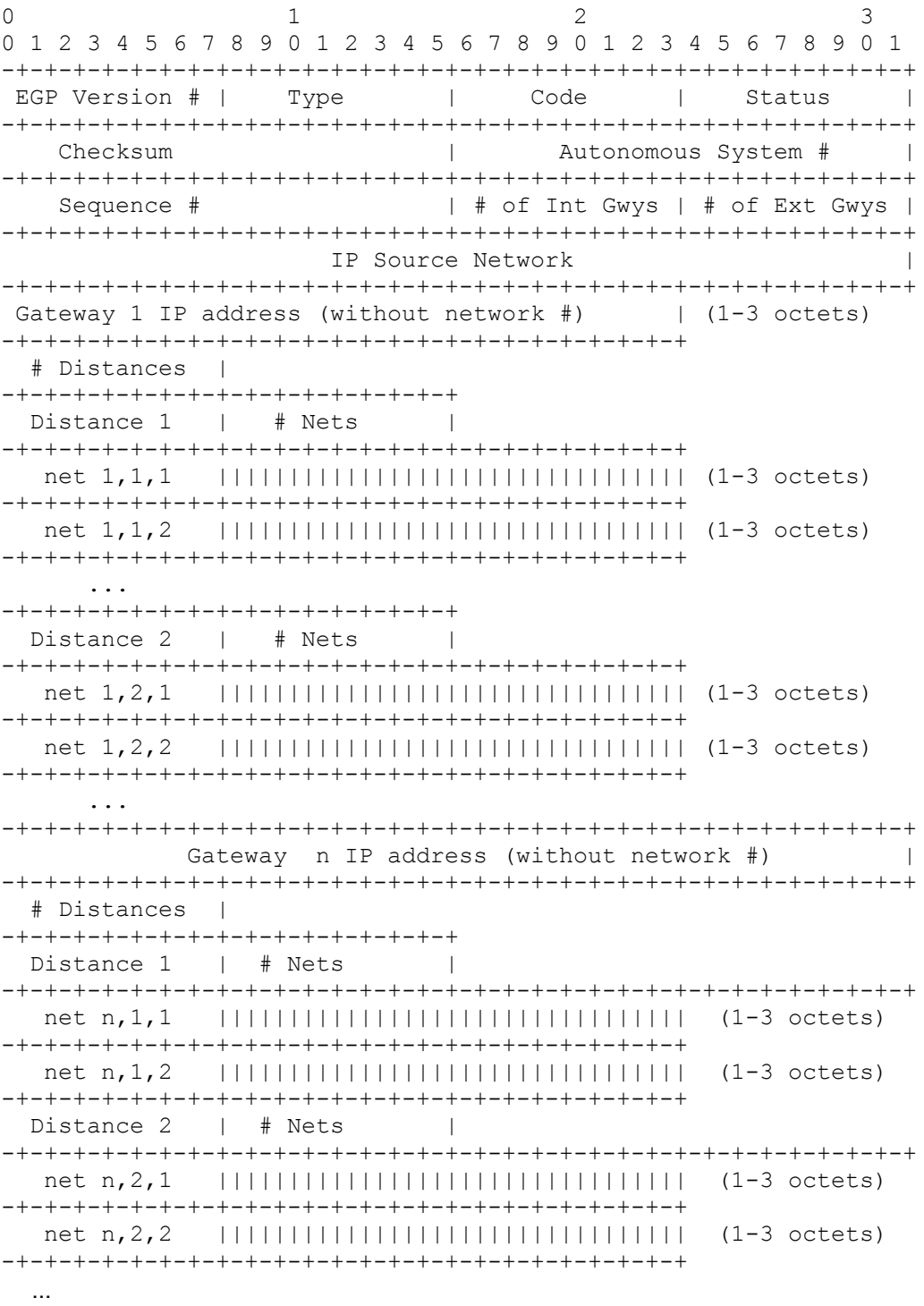
Type	5	
Code	0	Hello command
	1	I-H-U response
Status	0	indeterminate
	1	Up state
	2	Down state

Poll Command



Type	2	
Code	0	
Status	0	indeterminate
	1	Up state
	2	Down state
IP Source Network	IP network number of the network about which reachability information is being requested (coded as 1, 2 or 3 octets, left justified with trailing zeros)	

Update Response/Indication



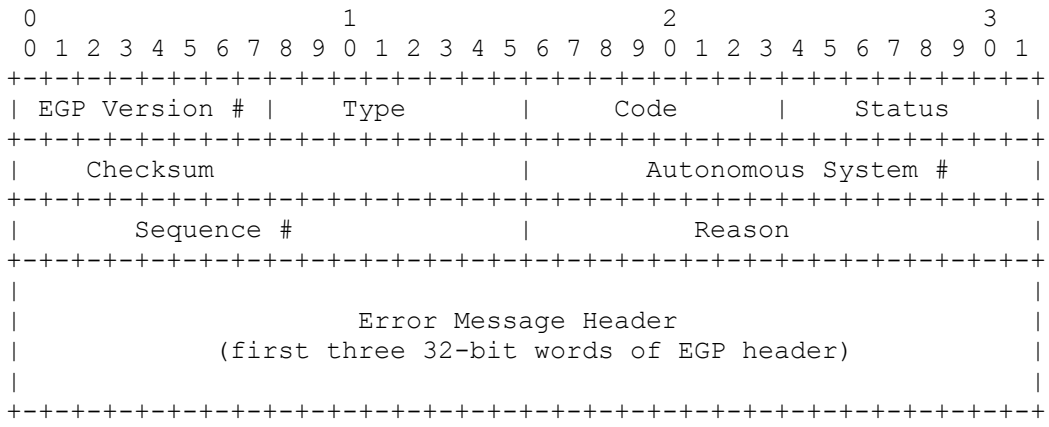
Type 1

Code 0

Status	0	indeterminate
	1	Up state
	2	Down state
	128	unsolicited message bit

# of Int Gwys	number of interior gateways appearing in this message
# of Ext Gwys	number of exterior gateways appearing in this message
IP Source Network	IP network number of the network about which reachability information is being supplied (coded as 1, 2 or 3 octets, left justified with trailing zeros)
Gateway IP addresses	IP address (without network number) of the gateway block (coded as 1, 2 or 3 octets)
# of Distances	number of distances in the gateway block
Distances	numbers depending on autonomous system architecture
# of Nets	number of nets at each distance
Nets	IP network number reachable via the gateway

Error Response/Indication



Type	8	
Code	0	
Status	0	indeterminate
	1	Up state
	2	Down state
	128	unsolicited message bit
Reason (see below)	0	unspecified
	1	bad EGP header format
	2	bad EGP data field format
	3	reachability info unavailable
	4	excessive polling rate
	5	no response
Error Message Header	first three 32-bit words of EGP header	

Following is a summary of the assigned Reason codes along with a list of scenarios in which they might be used.

Code	**Reason**	**Scenarios**
0	unspecified	when nothing else fits
1	bad EGP header format	1. bad message length 2. invalid Type, Code or Status fields Notes: The recipient can determine which of the above hold by inspecting the EGP header included in the message. An instance of a wrong EGP version or bad checksum should not be reported, since the original recipient can not trust the header format. An instance of an unknown autonomous system should be caught at acquisition time.
2	bad EGP data field format	1. nonsense polling rates (Request/Confirm)

		2. invalid Update message format 3. response IP Net Address field does not match command (Update) Notes: An instance of nonsense polling intervals (e.g. too long to be useful) specified in a Request or Confirm should result in a Refuse or Cease with this cause specified.
3	reachability info unavailable	1. no info available on net specified in IP Net Address field (Poll)
4	excessive polling rate	1. two or more Hello commands received within minimum specified polling interval 2. two or more Poll commands received within minimum specified polling interval 3. two or more Request commands received within some (reasonably short) interval Notes: The recipient can determine which of the above hold by inspecting the EGP header included in the message.
5	no response	1. no Update received for Poll within some (reasonably long) interval

Appendix B. Comparison with RFC-888

Minor functional enhancements are necessary in the RFC-888 message formats to support certain features assumed of the state-machine model, in particular the capability to request a neighbor to suppress Hello commands. In addition, the model suggests a mapping between its states and certain status and error indications which clarifies and generalizes the interpretation.

All of the header fields except the Status field (called the Information field at some places in RFC-888) remain unchanged. The following table summarizes the suggested format changes in the Status field for the various messages by (Type, Code) class.

Class	Messages	Status Codes	
3,0	Request	0	unspecified
3,1	Confirm	1	active mode
3,2	Refuse	2	passive mode
3,3	Cease	3	insufficient resources
3,4	Cease-ack	4	administratively prohibited
		5	going down
		6	parameter problem
5,0	Hello	0	indeterminate
5,1	I-H-U	1	Up state
2,0	Poll	2	Down state
1,0	Update	128	unsolicited message bit
8,0	Error		

The changes from RFC-888 are as follows:

1. The status codes have been combined in two classes, one for those messages involved in starting and stopping the protocol and the other for those messages involved in maintaining the protocol and exchanging reachability information. Some messages of either class may not use all the status codes assigned.

2. The status codes for the Request and Confirm indicate whether the sender can operate in active or passive mode. In RFC-888 this field must be zero; however, RFC-888 does not specify any mechanism to decide how the neighbors poll each other.

3. The status codes for the Cease, Refuse and Cease-ack have the same interpretation. This provides a clear and unambiguous indication when the protocol is terminated due to an unusual situation, for instance if the NOC dynamically repartitions the ARPANET. The assigned codes are not consistent with RFC-888, since the codes for the Refuse and Cease were assigned conflicting values; however, the differences are minor and should cause no significant problems.

4. The status codes for the Hello, I-H-U, Poll, Update and Error have the same interpretation. Codes 0 through 2 are mutually exclusive and are chosen solely on the basis of the state of the sender. In the case of the Update (and possibly Error) one of these codes can be combined with the "unsolicited bit," which corresponds to code 128. In RFC-888 this field is unused for the Poll and Error and may contain only zero or 128 for the Update, so that the default case is to assume that reciprocal reachability cannot be determined by these messages.

5. Some of the reachability codes defined in RFC-888 have been removed as not applicable.

Appendix C. Reachability Analysis

The following table shows the state transitions which can occur in a system of two neighboring EGP gateways. Besides being useful in the design and verification of the protocol, the table is useful for implementation and testing.

The system of two neighboring EGP gateways is modelled as a finite-state automaton constructed as the Cartesian product of two state machines as defined above. Each state of this machine is represented as [i,j], where i and j are states of the original machine. Each line of the table shows one state transition of the machine in the form:

[i1,j1] -> [i2,j2] E A

which specifies the machine in state [i1,j1] presented with event E transitions to state [i2,j2] and generates action A. Multiple actions are separated by the "/" symbol. The special symbol "*" represents the set of lines where all "*"s in the line take on the (same) values 0 - 4 in turn.

The table shows only those transitions which can occur as the result of events arriving at one of the two neighbors. The full table includes a duplicate set of lines for the other neighbor as well, with each line derived from a line of the table below using the transformation:

[i1,j1] -> [i2,j2] E A => [j1,i1] -> [j2,i2] E A

State	State	Event	Actions
[*,4] ->	[0,4]	Cease	Cease-ack
[0,1] ->	[2,1]	Request	Confirm/Hello/Up/t1
[0,1] ->	[0,1]	Request	Refuse
[0,*] ->	[1,*]	Start	Request/t1
[1,1] ->	[2,1]	Request	Confirm/Hello/Up/t1
[1,2] ->	[2,2]	Confirm	Hello/Up/t1
[1,3] ->	[2,3]	Confirm	Hello/Up/t1
[1,0] ->	[0,0]	Refuse	Null
[1,*] ->	[1,*]	Start	Request/r1
[1,*] ->	[0,*]	Stop	Null
[1,*] ->	[1,*]	t1	Request/t1
[2,1] ->	[3,1]	Up	Down/Hello/Poll/t1/t2
[2,1] ->	[2,1]	Request	Confirm/Hello/Up/t1
[2,2] ->	[2,2]	Hello	I-H-U
[2,3] ->	[2,3]	Hello	I-H-U
[2,2] ->	[2,2]	I-H-U	Process

```
[2,3] -> [2,3]  I-H-U          Process
[2,*] -> [1,*]  Start          Request/r1
[2,*] -> [4,*]  Stop           Cease/t1
[2,1] -> [2,1]  t1             Hello/t1
[2,2] -> [2,2]  t1             Hello/t1
[2,3] -> [2,3]  t1             Hello/t1

[3,1] -> [2,1]  Down           Null
[3,2] -> [2,2]  Down           Null
[3,3] -> [2,3]  Down           Null
[3,1] -> [2,1]  Request        Confirm/Hello/Up/t1
[3,2] -> [3,2]  Hello          I-H-U
[3,3] -> [3,3]  Hello          I-H-U
[3,2] -> [3,2]  I-H-U          Process
[3,3] -> [3,3]  I-H-U          Process
[3,3] -> [3,3]  Poll           Update
[3,3] -> [3,3]  Update         Process
[3,*] -> [1,*]  Start          Request/r1
[3,*] -> [4,*]  Stop           Cease/t1
[3,1] -> [3,1]  t1             Hello/t1
[3,2] -> [3,2]  t1             Hello/t1
[3,3] -> [3,3]  t1             Hello/t1
[3,1] -> [3,1]  t2             Poll/t2
[3,2] -> [3,2]  t2             Poll/t2
[3,3] -> [3,3]  t2             Poll/t2

[4,1] -> [4,1]  Request        Cease
[4,*] -> [0,*]  Cease          Cease-ack
[4,0] -> [0,0]  Cease-ack      Null
[4,*] -> [0,*]  Stop           Null
[4,*] -> [4,*]  t1             Cease/t1
```

In the state-machine model defined in this document all states of the above machine are reachable; however, some are reachable only in extreme cases when one neighbor crashes, for example. In the common case where only one of the neighbors initiates and terminates the protocol and neither one crashes, for example, not all states are reachable. Following is a matrix showing the states which can be reached in this case, where the neighbor that initiates and terminates the protocol is called the active gateway and the other the passive gateway.

Active Gateway	Passive Gateway: 0 Idle	1 Aqsn	2 Down	3 Up	4 Cease
0 Idle	stable				unstable
1 Aqsn	unstable	unstable	unstable	unstable	unstable
2 Down			stable	unstable	
3 Up			unstable	stable	
4 Cease	unstable	unstable	unstable	unstable	unstable

In the above matrix the blank entries represent unreachable states, while those marked unstable represent transient states which cannot persist for long, due to retransmission of Request and Hello messages, for example.

RFC-906 Bootstrap Loading using TFTP

Ross Finlayson
Stanford University
June 1984

Status of this Memo

It is often convenient to be able to bootstrap a computer system from a communications network. This RFC proposes the use of the IP Trivial File Transfer Protocol for bootstrap loading in this case.

Acknowledgements

The ideas presented here are the result of discussions with several other people, in particular Jeff Mogul.

RFC-906 Bootstrap Loading using TFTP

Introduction

Many computer systems, such as diskless workstations, are bootstrapped by loading one or more code files across a network. Unfortunately, the protocol used to load these initial files has not been standardized - numerous methods have been employed by different computer manufacturers. This can make it difficult, for example, for an installation to support several different kinds of systems on a local-area network. Each different booting mechanism that is used must be supported, for example by implementing a number of servers on one or more host machines. This is in spite of the fact that these heterogeneous systems may be able to communicate freely (all using the same protocol) once they have been booted.

We propose that TFTP (Trivial File Transfer Protocol) [RFC-783] be used as a standard protocol for bootstrap loading. This protocol is well-suited for our purpose, being based on the standard Internet Protocol (IP) [RFC-791]. It is easily implemented, both in the machines to be booted, and in bootstrap servers elsewhere on the net. (In addition, many popular operating systems already support TFTP servers.) The fact that TFTP is a rather slow protocol is not a serious concern, due to the fact that it need be used only for the primary bootstrap. A secondary bootstrap could use a faster protocol.

This RFC describes how system to be booted (called the "booter" below) would use TFTP to load a desired code file. It also describes an existing implementation (in ROM) for Ethernet.

Note that we are specifying only the network protocols that would be used by the booting system. We do not attempt to mandate the method by which a user actually boots a system (such as the format of a command typed at the console). In addition, our proposal does not presuppose the use of any particular data-link level network architecture (although the example that we describe below uses Ethernet).

RFC-906 Bootstrap Loading using TFTP

Network Protocols used by the Booting System

To load a file, the booter sends a standard TFTP read request (RRQ) packet, containing the name of the file to be loaded. The file name should not assume any operating system dependent naming conventions (file names containing only alphanumeric characters should suffice). Thereafter, the system receives TFTP DATA packets, and sends TFTP ACK and/or ERROR packets, in accordance with the TFTP specification [RFC-783].

TFTP is implemented using the User Datagram Protocol (UDP) [RFC-768], which is in turn implemented using IP. Thus, the booter must be able to receive IP datagrams containing up to 524 octets (excluding the IP header), since TFTP DATA packets can be up to 516 octets long, and UDP headers are 8 octets long. The booting machine is not required to respond to incoming TFTP read or write requests.

We allow for the use of two additional protocols. These are ARP (Address Resolution Protocol) [RFC-826], and RARP (Reverse Address Resolution Protocol) [RFC-903]. The possible use of these protocols is described below. The booter could also use other protocols (such as for name lookup), but they should be IP-based, and an internet standard.

The IP datagram containing the initial TFTP RRQ (and all other IP datagrams sent by the booter) must of course contain both a source internet address and a destination internet address in its IP header. It is frequently the case, however, that the booter does not initially know its own internet address, but only a lower-level (e.g. Ethernet) address. The Reverse Address Resolution Protocol (RARP) [RFC-903] may be used by the booter to find its internet address (prior to sending the TFTP RRQ). RARP was motivated by Plummer's Address Resolution Protocol (ARP) [RFC-826]. Unlike ARP, which is used to find the 'hardware' address corresponding to a known higher-level protocol (e.g. internet) address, RARP is used to determine a higher-level protocol address, given a known hardware address. RARP uses the same packet format as ARP, and like ARP, can be used for a wide variety of data-link protocols.

ARP may also be used. If the destination internet address is known, then an ARP request containing this address may be broadcast, to find a corresponding hardware address to which to send the subsequent TFTP RRQ. It may not matter if this request should fail, because the RRQ can also be broadcast (at the data-link level). However, because such an ARP request packet also contains the sender's (that is, the booter's) internet and hardware addresses, this information is made available to the rest of the local subnet, and could be useful for routing, for instance.

If a single destination internet address is not known, then a special 'broadcast' internet address could be used as the destination address in the TFTP RRQ, so that it will be received by all 'local' internet hosts.

An Example Implementation

The author has implemented TFTP booting as specified above. The resulting code resides in ROM. (This implementation is for a Motorola 68000 based workstation, booting over an Ethernet.) A user wishing to boot such a machine types a file name, and (optionally) the internet address of the workstation, and/or the internet address of a server machine from which the file is to be loaded. The bootstrap code proceeds as follows:

(1) The workstation's Ethernet address is found (by querying the Ethernet interface).

(2) If the internet address of the workstation was not given, then a RARP request is broadcast, in order to find it. If this request fails (that is, times out), then the bootstrap fails.

(3) If the internet address of a server host was given, then broadcast an ARP request to try to find a corresponding Ethernet address. If this fails, or if a server internet address was not given, then the Ethernet broadcast address is used.

(4) If the internet address of a server host was not given, then we use a special internet address that represents a broadcast on the "local subnet", as described in [2]. (This is not an internet standard.)

(5) A TFTP RRQ for the requested file is sent to the Ethernet address found in step (3). The source internet address is that found in step (2), and the destination internet address is that found in step (4).

Note that because several TFTP servers may, in general, reply to the RRQ, we do not abort if a TFTP ERROR packet is received, because this does not preclude the possibility of some other server replying later with the first data packet of the requested file. When the first valid TFTP DATA packet is received in response to the RRQ, the source internet and Ethernet addresses of this packet are used as the destination addresses in subsequent TFTP ACK packets. Should another server later respond with a DATA packet, an ERROR packet is sent back in response.

An implementation of TFTP booting can take up a lot of space if care is not taken. This can be a significant problem if the code is to fit in a limited amount of ROM. However, the implementation described above consists of less than 4K bytes of code (not counting the Ethernet device driver).

RFC-917 Internet Subnets

Jeffrey Mogul
Computer Science Department
Stanford University
October 1984

Status Of This Memo

This RFC suggests a proposed protocol for the ARPA-Internet community, and requests discussion and suggestions for improvements. Distribution of this memo is unlimited. The protocol described here is largely superceded by "Internet Standard Subnetting" [RFC-950].

RFC-917 Internet Subnets

Introduction

Overview

We discuss the utility of "subnets" of Internet networks, which are logically visible sub-sections of a single Internet network. For administrative or technical reasons, many organizations have chosen to divide one Internet network into several subnets, instead of acquiring a set of Internet network numbers.

We propose procedures for the use of subnets, and discuss approaches to solving the problems that arise, particularly that of routing.

Acknowledgment

This proposal is the result of discussion with several other people. J. Noel Chiappa, Chris Kent, and Tim Mann, in particular, provided important suggestions.

Introduction

The original view of the Internet universe was a two-level hierarchy: the top level the catenet as a whole, and the level below it a collection of "Internet Networks", each with its own Network Number. (We do not mean that the Internet has a hierarchical topology, but that the interpretation of addresses is hierarchical.)

While this view has proved simple and powerful, a number of organizations have found it inadequate and have added a third level to the interpretation of Internet addresses. In this view, a given Internet Network might (or might not) be divided into a collection of subnets.

The original, two-level, view carries a strong presumption that, to a host on an Internet network, that network may be viewed as a single edge; to put it another way, the network may be treated as a "black box" to which a set of hosts is connected. This is true of the ARPANET, because the IMPs mask the use of specific links in that network. It is also true of most local area network (LAN) technologies, such as Ethernet or ring networks.

However, this presumption fails in many practical cases, because in moderately large organizations (e.g., Universities or companies with more than one building) it is often necessary to use more than one LAN cable to cover a "local area". For example, at this writing there are eighteen such cables in use at Stanford University, with more planned.

There are several reasons why an organization might use more than one cable to cover a campus:

- Different technologies: Especially in a research environment, there may be more than one kind of LAN in use; e.g., an organization may have some equipment that supports Ethernet, and some that supports a ring network.
- Limits of technologies: Most LAN technologies impose limits, based electrical parameters, on the number of hosts connected, and on the total length of the cable. It is easy to exceed these limits, especially those on cable length.
- Network congestion: It is possible for a small subset of the hosts on a LAN to monopolize most of the bandwidth. A common solution to this problem is to divide the hosts into cliques of high mutual communication, and put these cliques on separate cables.
- Point-to-Point links: Sometimes a "local area", such as a university campus, is split into two locations too far apart to connect using the preferred LAN technology. In this case, high-speed point-to-point links might connect

several LANs.

An organization that has been forced to use more than one LAN has three choices for assigning Internet addresses:

1. Acquire a distinct Internet network number for each cable.
2. Use a single network number for the entire organization, but assign host numbers without regard to which LAN a host is on. (We will call this choice "transparent subnets".)
3. Use a single network number, and partition the host address space by assigning subnet numbers to the LANs. ("Explicit subnets".)

Each of these approaches has disadvantages. The first, although not requiring any new or modified protocols, does result in an explosion in the size of Internet routing tables. Information about the internal details of local connectivity is propagated everywhere, although it is of little or no use outside the local organization. Especially as some current gateway implementations do not have much space for routing tables, it would be nice to avoid this problem.

The second approach requires some convention or protocol that makes the collection of LANs appear to be a single Internet network. For example, this can be done on LANs where each Internet address is translated to a hardware address using an Address Resolution Protocol (ARP), by having the bridges between the LANs intercept ARP requests for non-local targets. However, it is not possible to do this for all LAN technologies, especially those where ARP protocols are not currently used, or if the LAN does not support broadcasts. A more fundamental problem is that bridges must discover which LAN a host is on, perhaps by using a broadcast algorithm. As the number of LANs grows, the cost of broadcasting grows as well; also, the size of translation caches required in the bridges grows with the total number of hosts in the network.

The third approach addresses the key problem: existing standards assume that all hosts on an Internet local network are on a single cable. The solution is to explicitly support subnets. This does have a disadvantage, in that it is a modification of the Internet Protocol, and thus requires changes to IP implementations already in use (if these implementations are to be used on a subnetted network.) However, we believe that these changes are relatively minor, and once made, yield a simple and efficient solution to the problem. Also, the approach we take in this document is to avoid any changes that would be incompatible with existing hosts on non-subnetted networks.

Further, when appropriate design choices are made, it is possible for hosts which believe they are on a non-subnetted network to be used on a subnetted one, as will be explained later. This is useful when it is not possible to modify some of the hosts to support subnets explicitly, or when a gradual transition is preferred. Because of this, there seems little reason to use the second approach listed above.

The rest of this document describes approaches to subnets of Internet Networks.

Terminology

To avoid either ambiguity or prolixity, we will define a few terms, which will be used in the following sections:

Catenet

The collection of connected Internet Networks

Network

A single Internet network (that may or may not be divided into subnets.)

Subnet

A subnet of an Internet network.

Network Number

As in [8].

Local Address

The bits in an Internet address not used for the network number; also known as "rest field".

Subnet Number

A number identifying a subnet within a network.

Subnet Field

The bit field in an Internet address used for the subnet number.

Host Field

The bit field in an Internet address used for denoting a specific host.

Gateway

A node connected to two or more administratively distinct networks and/or subnets, to which hosts send datagrams to be forwarded.

Bridge

A node connected to two or more administratively indistinguishable but physically distinct subnets, that automatically forwards datagrams when necessary, but whose existence is not know to other hosts. Also called a "software repeater".

Standards for Subnet Addressing

Following the division presented in [2], we observe that subnets are fundamentally an issue of addressing. In this section, we first describe a proposal for interpretation of Internet Addressing to support subnets. We then discuss the interaction between this address format and broadcasting; finally, we present a protocol for discovering what address interpretation is in use on a given network.

RFC-917 Internet Subnets - Standards for Subnet Addressing

Interpretation of Internet Addresses

Suppose that an organization has been assigned an Internet network number, has further divided that network into a set of subnets, and wants to assign host addresses: how should this be done? Since there are minimal restrictions on the assignment of the "local address" part of the Internet address, several approaches have been proposed for representing the subnet number:

1. Variable-width field: Any number of the bits of the local address part are used for the subnet number; the size of this field, although constant for a given network, varies from network to network. If the field width is zero, then subnets are not in use.
2. Fixed-width field: A specific number of bits (e.g., eight) is used for the subnet number, if subnets are in use.
3. Self-encoding variable-width field: Just as the width (i.e., class) of the network number field is encoded by its high-order bits, the width of the subnet field is similarly encoded.
4. Self-encoding fixed-width field: A specific number of bits is is used for the subnet number. Subnets are in use if the high-order bit of this field is one; otherwise, the entire local address part is used for host number.

Since there seems to be no advantage in doing otherwise, all these schemes place the subnet field as the most significant field in the local address part. Also, since the local address part of a Class C address is so small, there is little reason to support subnets of other than Class A and Class B networks.

What criteria can we use to choose one of these four schemes? First, do we want to use a self-encoding scheme; that is, should it be possible to tell from examining an Internet address if it refers to a subnetted network, without reference to any other information?

One advantage to self-encoding is that it allows one to determine if a non-local network has been divided into subnets. It is not clear that this would be of any use. The principle advantage, however, is that no additional information is needed for an implementation to determine if two addresses are on the same subnet. However, this can also be viewed as a disadvantage: it may cause problems for non-subnetted networks which have existing host numbers that use arbitrary bits in the local address part <1>. In other words, it is useful to be able control whether a network is subnetted independently from the assignment of host addresses. Another disadvantage of any self-encoding scheme is that it reduces the local address space by at least a factor of two.

If a self-encoding scheme is not used, it is clear that a variable-width subnet field is appropriate. Since there must in any case be some per-network "flag" to indicate if subnets are in use, the additional cost of using an integer (the subnet field width) instead of a boolean is negligible. The advantage of using a variable-width subnet field is that it allows each organization to choose the best way to allocate relatively scarce bits of local address to subnet and host numbers.

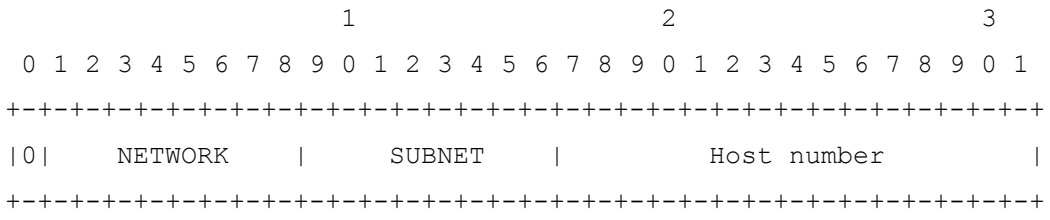
Our proposal, therefore, is that the Internet address be interpreted as:

<network-number><subnet-number><host-number>

where the <network-number> field is as in [8], the <host-number> field is at least one bit wide, and the width of the <subnet-number> field is constant for a given network. No further structure is required for the <subnet-number> or <host-number> fields. If the width of the <subnet-number> field is zero, then the network is not subnetted (i.e., the

interpretation of [8] is used.)

For example, on a Class A network with an eight bit wide subnet field, an address is broken down like this:



We expect that, for reasons of simplicity and efficient implementation, that most organizations will choose a subnet field width that is a multiple of eight bits. However, an implementation must be prepared to handle other possible widths.

We reject the use of "recursive subnets", the division of the host field into "sub-subnet" and host parts, because:

- There is no obvious need for a four-level hierarchy.
- The number of bits available in an IP address is not large enough to make this useful in general.
- The extra mechanism required is complex.

Changes to Host Software to Support Subnets

In most implementations of IP, there is code in the module that handles outgoing packet that does something like:

```
IF ip_net_number(packet.ip_dest) = ip_net_number(my_ip_addr)
    THEN
        send_packet_locally(packet, packet.ip_dest)
    ELSE
        send_packet_locally(packet,
            gateway_to(ip_net_number(packet.ip_dest)))
```

(If the code supports multiple connected networks, it will be more complicated, but this is irrelevant to the current discussion.)

To support subnets, it is necessary to store one more 32-bit quantity, called my_ip_mask. This is a bit-mask with bits set in the fields corresponding to the IP network number, and additional bits set corresponding to the subnet number field. For example, on a Class A network using an eight-bit wide subnet field, the mask would be 255.255.0.0.

The code then becomes:

```
IF bitwise_and(packet.ip_dest, my_ip_mask)
                = bitwise_and(my_ip_addr, my_ip_mask)
    THEN
        send_packet_locally(packet, packet.ip_dest)
    ELSE
        send_packet_locally(packet,
            gateway_to(bitwise_and(packet.ip_dest, my_ip_mask)))
```

Of course, part of the expression in the conditionally can be pre-computed.

It may or may not be necessary to modify the "gateway_to" function, so that it performs comparisons in the same way.

To support multiply-connected hosts, the code can be changed to keep the "my_ip_addr" and "my_ip_mask" quantities on a per-interface basis; the expression in the conditional must then be evaluated for each interface.

Subnets and Broadcasting

In the absence of subnets, there are only two kinds of broadcast possible within the Internet Protocol <2>: broadcast to all hosts on a specific network, or broadcast to all hosts on "this network"; the latter is useful when a host does not know what network it is on.

When subnets are used, the situation becomes slightly more complicated. First, the possibility now exists of broadcasting to a specific subnet. Second, broadcasting to all the hosts on a subnetted network requires additional mechanism; in [6] the use of "Reverse Path Forwarding" [3] is proposed. Finally, the interpretation of a broadcast to "this network" is that it should not be forwarded outside of the original subnet.

Implementations must therefore recognize three kinds of broadcast addresses, in addition to their own host addresses:

This physical network
A destination address of all ones (255.255.255.255) causes the a datagram to be sent as a broadcast on the local physical network; it must not be forwarded by any gateway.

Specific network
The destination address contains a valid network number; the local address part is all ones (e.g., 36.255.255.255).

Specific subnet
The destination address contains a valid network number and a valid subnet number; the host field is all ones (e.g., 36.40.255.255).

For further discussion of Internet broadcasting, see [6].

One factor that may aid in deciding whether to use subnets is that it is possible to broadcast to all hosts of a subnetted network with a single operation at the originating host. It is not possible to broadcast, in one step, to the same set of hosts if they are on distinct networks.

Determining the Width of the Subnet Field

How can a host (or gateway) determine what subnet field width is in use on a network to which it is connected? The problem is analogous to several other "bootstrapping" problems for Internet hosts: how a host determines its own address, and how it locates a gateway on its local network. In all three cases, there are two basic solutions: "hardwired" information, and broadcast-based protocols.

"Hardwired" information is that available to a host in isolation from a network. It may be compiled-in, or (preferably) stored in a disk file. However, for the increasingly common case of a diskless workstation that is bootloaded over a LAN, neither hard-wired solution is satisfactory. Instead, since most LAN technology supports broadcasting, a better method is for the newly-booted host to broadcast a request for the necessary information. For example, for the purpose of determining its Internet address, a host may use the "Reverse Address Resolution Protocol" [4].

We propose to extend the ICMP protocol [9] by adding a new pair of ICMP message types, "Address Format Request" and "Address Format Reply", analogous to the "Information Request" and "Information Reply" ICMP messages. These are described in detail in Appendix I.

The intended use of these new ICMPs is that a host, when booting, broadcast an "Address Format Request" message <3>. A gateway (or a host acting in lieu of a gateway) that receives this message responds with an "Address Format Reply". If there is no indication in the request which host sent it (i.e., the IP Source Address is zero), the reply is broadcast as well. The requesting host will hear the response, and from it determine the width of the subnet field.

Since there is only one possible value that can be sent in an "Address Format Reply" on any given LAN, there is no need for the requesting host to match the responses it hears against the request it sent; similarly, there is no problem if more than one gateway responds. We assume that hosts reboot infrequently, so the broadcast load on a network from use of this protocol should be small.

If a host is connected to more than one LAN, it must use this protocol on each, unless it can determine (from a response on one of the LANs) that several of the LANs are part of the same network, and thus must have the same subnet field width.

One potential problem is what a host should do if it receives no response to its "Address Format Request", even after a reasonable number of tries. Three interpretations can be placed on the situation:

1. The local net exists in (permanent) isolation from all other nets.
2. Subnets are not in use, and no host supports this ICMP request.
3. All gateways on the local net are (temporarily) down.

The first and second situations imply that the subnet field width is zero. In the third situation, there is no way to determine what the proper value is; the safest choice is thus zero. Although this might later turn out to be wrong, it will not prevent transmissions that would otherwise succeed. It is possible for a host to recover from a wrong choice: when a gateway comes up, it should broadcast an "Address Format Reply"; when a host receives such a message that disagrees with its guess, it should adjust its data structures to conform to the received value. No host or gateway should send an "Address Format Reply" based on a "guessed" value.

Finally, note that no host is required to use this ICMP protocol to discover the subnet field

width; it is perfectly reasonable for a host with non-volatile storage to use stored information.

Subnet Routing Methods

One problem that faces all Internet hosts is how to determine a route to another host. In the presence of subnets, this problem is only slightly modified.

The use of subnets means that there are two levels to the routing process, instead of one. If the destination host is on the same network as the source host, the routing decision involves only the subnet gateways between the hosts. If the destination is on a different network, then the routing decision requires the choice both of a gateway out of the source host's network, and of a route within the network to that gateway.

Fortunately, many hosts can ignore this distinction (and, in fact, ignore all routing choices) by using a "default" gateway as the initial route to all destinations, and relying on ICMP Host Redirect messages to define more appropriate routes. However, this is not an efficient method for a gateway or for a multi-homed host, since a redirect may not make up for a poor initial choice of route. Such hosts should use a routing information exchange protocol, but that is beyond the scope of this document; in any case, the problem arises even when subnets are not used.

The problem for a singly-connected host is thus to find at least one neighbor gateway. Again, there are basic two solutions to this: use hard-wired information, or use broadcasts. We believe that the neighbor-gateway acquisition problem is the same with or without subnets, and thus the choice of solution is not affected by the use of subnets.

However, one problem remains: a source host must determine if datagram to a given destination address must be sent via a gateway, or sent directly to the destination host. In other words, is the destination host on the same physical network as the source? This particular phase of the routing process is the only one that requires an implementation to be explicitly aware of subnets; in fact, if broadcasts are not used, it is the only place where an Internet implementation must be modified to support subnets.

Because of this, it is possible to use some existing implementations without modification in the presence of subnets <4>. For this to work, such implementations must:

- Be used only on singly-homed hosts, and not as a gateway.
- Be used on a broadcast LAN.
- Use an Address Resolution Protocol (ARP), such [7].
- Not be required to maintain connections in the case of gateway crashes.

In this case, one can modify the ARP server module in a subnet gateway so that when it receives an ARP request, it checks the target Internet address to see if it is along the best route to the target. If it is, it sends to the requesting host an ARP response indicating its own hardware address. The requesting host thus believes that it knows the hardware address of the destination host, and sends packets to that address. In fact, the packets are received by the gateway, and forwarded to the destination host by the usual means.

This method requires some blurring of the layers in the gateways, since the ARP server and the Internet routing table would normally not have any contact. In this respect, it is somewhat unsatisfactory. Still, it is fairly easy to implement, and does not have significant performance costs. One problem is that if the original gateway crashes, there is no way for the source host to choose an alternate route even if one exists; thus, a connection that might otherwise have been maintained will be broken.

One should not confuse this method of "ARP-based subnetting" with the superficially similar use of ARP-based bridges. ARP-based subnetting is based on the ability of a gateway to

examine an IP address and deduce a route to the destination, based on explicit subnet topology. In other words, a small part of the routing decision has been moved from the source host into the gateway. An ARP-based bridge, in contrast, must somehow locate each host without any assistance from a mapping between host address and topology. Systems built out of ARP-based bridges should not be referred to as "subnetted".

N.B.: the use of ARP-based subnetting is complicated by the use of broadcasts. An ARP server [7] should never respond to a request whose target is a broadcast address. Such a request can only come from a host that does not recognize the broadcast address as such, and so honoring it would almost certainly lead to a forwarding loop. If there are N such hosts on the physical network that do not recognize this address as a broadcast, then a packet sent with a Time-To-Live of T could potentially give rise to T**N spurious re-broadcasts.

Stanford University

At Stanford, subnets were introduced initially for historical reasons. Stanford had been using the Pup protocols [1] on a collection of several Experimental Ethernets [5] since 1979, several years before Internet protocols came into use. There were a number of Pup gateways in service, and all hosts and gateways acquired and exchanged routing table information using a simple broadcast protocol.

When the Internet Protocol was introduced, the decision was made to use an eight-bit wide subnet number; Internet subnet numbers were chosen to match the Pup network number of a given Ethernet, and the Pup host numbers (also eight bits) were used as the host field of the Internet address.

The Pup-only gateways were then modified to forward Internet datagrams according to their Pup routing tables; they otherwise had no understanding of Internet packets and in fact did not adjust the Time-to-live field in the Internet header. This seems to be acceptable, since bugs that caused forwarding loops have not appeared. The Internet hosts that are multi-homed and thus can serve as gateways do adjust the Time-to-live field; since all of the currently also serve as Pup gateways, no additional routing information exchange protocol was needed.

Internet host implementations were modified to understand subnets (in several different ways, but with identical effects). Since all already had Pup implementations, the Internet routing tables were maintained by the same process that maintained the Pup routing tables, simply translating the Pup network numbers into Internet subnet numbers.

When 10Mbit Ethernets were added, the gateways were modified to use the ARP-based scheme described in an earlier section; this allowed unmodified hosts to be used on the 10Mbit Ethernets.

IP subnets have been in use since early 1982; currently, there are about 330 hosts, 18 subnets, and a similar number of subnet gateways in service. Once the Pup-only gateways are converted to be true Internet gateways, an Internet-based routing exchange protocol will be introduced, and Pup will be phased out.

MIT

MIT was the first IP site to accumulate a large collection of local network links. Since this happened before network numbers were divided into classes, to have assigned each link at MIT its own IP network number would have used up a good portion of the available address space. MIT decided to use one IP network number, and to manage the 24-bit "rest" field itself, by dividing it into three 8-bit fields; "subnet", "reserved, must be zero", and "host". Since the CHAOS protocol already in use at MIT used an 8-bit subnet number field, it was possible to assign each link the same subnet number in both protocols. The IP host field was set to 8 bits since most available local net hardware at that point used 8 bit addresses, as did the CHAOS protocol; it was felt that reserving some bits for the future was wise.

The initial plan was to use a dynamic routing protocol between the IP subnet gateways; several such protocols have been mooted but nobody has bothered to implement one; static routing tables are still used. It is likely that this change will finally be made soon.

To solve the problem that imported IP software always needed modification to work in the subnetted environment, MIT searched for a model of operation that led to the least change in host IP software. This led to a model where IP gateways send ICMP Host Redirects rather than Network Redirects. All internal MIT IP gateways now do so. With hosts that can maintain IP routing tables for non-local communication on a per host basis, this hides most of the subnet structure. The "minimum adjustment" for host software to work correctly in both subnetted and non-subnetted environments is the bit-mask algorithm mentioned earlier.

MIT has no immediate plans to move toward a single "approved" protocol; this is due partly to the degree of local autonomy and the amount of installed software, and partly to the lack of a single prominent industry standard. Rather, the approach taken has been to provide a single set of physical links and packet switches, and to layer several "virtual" protocol nets atop the single set of links. MIT has had some bad experiences with trying to exchange routing information between protocols and wrap one protocol in another; the general approach is to keep the protocols strictly separated except for sharing the basic hardware. Using ARP to hide the subnet structure is not much in favor; it is felt that this overloads the address resolution operation. In a complicated system (i.e. one with loops, and variant link speeds), a more sophisticated information interchange will be needed between gateways; making this an explicit mechanism (but one insulated from the hosts) was felt to be best.

RFC-917 Internet Subnets - Case Studies

Carnegie-Mellon University

CMU uses a Class B network currently divided into 11 physical subnets (two 3Mbit Experimental Ethernets, seven 10Mbit Ethernets, and two ProNet rings.) Although host numbers are assigned so that all addresses with a given third octet will be on the same subnet (but not necessarily vice versa), this is essentially an administrative convenience. No software currently knows the specifics of this allocation mechanism or depends on it to route between cables.

Instead, an ARP-based bridge scheme is used. When a host broadcasts an ARP request, all bridges which receive it cache the original protocol address mapping and then forward the request (after the appropriate adjustments) as an ARP broadcast request onto each of their other connected cables. When a bridge receives a non-broadcast ARP reply with a target protocol address not its own, it consults its ARP cache to determine the cable onto which the reply should be forwarded. The bridges thus attempt to transparently extend the ARP protocol into a heterogenous multi-cable environment. They are therefore required to turn ARP broadcasts on a single cable into ARP broadcasts on all other connected cables even when they "know better". This algorithm works only in the absence of cycles in the network connectivity graph (which is currently the case). Work is underway to replace this simple-minded algorithm with a protocol implemented among the bridges, in support of redundant paths and to reduce the collective broadcast load. The intent is to retain the ARP base and host transparency, if possible.

Implementations supporting the 3Mbit Ethernet and 10Mb proNET ring at CMU use RFC-826 ARP (instead of some wired-in mapping such as simply using the 8-bit hardware address as the the fourth octet of the IP address).

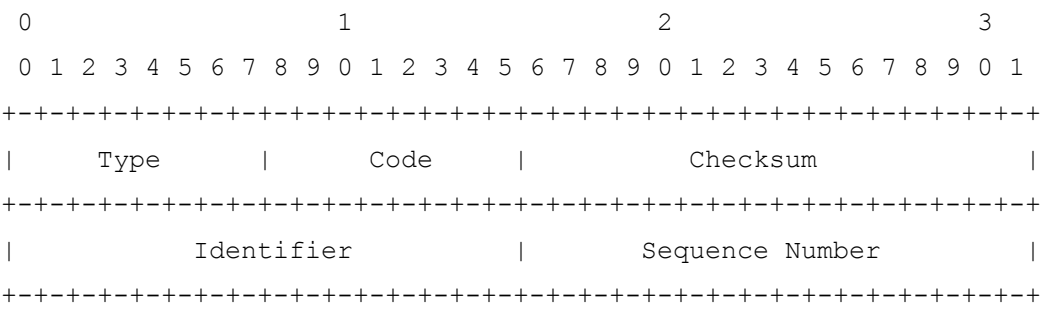
Since there are currently no redundant paths between cables, the issue of maintaining connections across bridge crashes is moot. With about 150 IP-capable hosts on the net, the bridge caches are still of reasonable size, and little bandwidth is devoted to ARP broadcast forwarding.

CMU's network is likely to grow from its relatively small, singly-connected configuration centered within their CS/RI facility to a campus-wide intra-departmental configuration with 5000-10000 hosts and redundant connections between cables. It is possible that the ARP-based bridge scheme will not scale to this size, and a system of explicit subnets may be required. The medium-term goal, however, is an environment into which unmodified extant (especially 10Mb ethernet based) IP implementations can be imported; the intent is to stay with a host-transparent (thus ARP-based) routing mechanism as long as possible. CMU is concerned that even if subnets become part of the IP standard they will not be widely implemented; this is the major obstacle to their use at CMU.

RFC-917 Internet Subnets

Address Format ICMP

Address Format Request or Address Format Reply



IP Fields:

Addresses

The address of the source in an address format request message will be the destination of the address format reply message. To form an address format reply message, the source address of the request becomes the destination address of the reply, the source address of the reply is set to the replier's address, the type code changed to A2, the subnet field width inserted into the Code field, and the checksum recomputed. However, if the source address in the request message is zero, then the destination address for the reply message should denote a broadcast.

ICMP Fields:

Type

A1 for address format request message

A2 for address format reply message

Code

0 for address format request message

Width of subnet field, in bits, for address format reply message

Checksum

The checksum is the 16-bit one's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum, the checksum field should be zero. This checksum may be replaced in the future.

Identifier

An identifier to aid in matching request and replies, may be zero.

Sequence Number

A sequence number to aid in matching request and replies, may be zero.

Description

A gateway receiving an address format request should return it with the Code field set to

the number of bits of Subnet number in IP addresses for the network to which the datagram was addressed. If the request was broadcast, the destination network is "this network". The Subnet field width may be from 0 to (31 - N), where N is the width in bits of the IP net number field (i.e., 8, 16, or 24).

If the requesting host does not know its own IP address, it may leave the source field zero; the reply should then be broadcast. Since there is only one possible address format for a network, there is no need to match requests with replies. However, this approach should be avoided if at all possible, since it increases the superfluous broadcast load on the network.

Type A1 may be received from a gateway or a host.

Type A2 may be received from a gateway, or a host acting in lieu of a gateway.

Examples

For these examples, we assume that the requesting host has address 36.40.0.123, that there is a gateway at 36.40.0.62, and that on network 36.0.0.0, an 8-bit wide subnet field is in use.

First, suppose that broadcasting is allowed, and that 36.40.0.123 knows its own address. It sends the following datagram:

Source address:	36.40.0.123
Destination address:	36.255.255.255
Protocol:	ICMP = 1
Type:	Address Format Request = A1
Code:	0

36.40.0.62 will hear the datagram, and should respond with this datagram:

Source address:	36.40.0.62
Destination address:	36.40.0.123
Protocol:	ICMP = 1
Type:	Address Format Reply = A2
Code:	8

For the following examples, assume that address 255.255.255.255 denotes "broadcast to this physical network", as described in [6].

The previous example is inefficient, because it potentially broadcasts the request on many subnets. The most efficient method, and the one we recommend, is for a host to first discover its own address (perhaps using the "Reverse ARP" protocol described in [4]), and then to send the ICMP request to 255.255.255.255:

Source address:	36.40.0.123
Destination address:	255.255.255.255
Protocol:	ICMP = 1
Type:	Address Format Request = A1
Code:	0

The gateway can then respond directly to the requesting host.

Suppose that 36.40.0.123 is a diskless workstation, and does not know even its own host number. It could send the following datagram:

Source address:	0.0.0.0
Destination address:	255.255.255.255
Protocol:	ICMP = 1
Type:	Address Format Request = A1
Code:	0

36.40.0.62 will hear the datagram, and should respond with this datagram:

Source address:	36.40.0.62
Destination address:	36.40.255.255
Protocol:	ICMP = 1
Type:	Address Format Reply = A2

Code:	8

Note that the gateway uses the narrowest possible broadcast to reply (i.e., sending the reply to 36.255.255.255 would mean that it is transmitted on many subnets, not just the one on which it is needed.) Even so, the overuse of broadcasts presents an unnecessary load to all hosts on the subnet, and so we recommend that use of the "anonymous" (0.0.0.0) source address be kept to a minimum.

If broadcasting is not allowed, we assume that hosts have wired-in information about neighbor gateways; thus, 36.40.0.123 might send this datagram:

Source address:	36.40.0.123
Destination address:	36.40.0.62
Protocol:	ICMP = 1
Type:	Address Format Request = A1
Code:	0

36.40.0.62 should respond exactly as in the previous case.

Notes

<1> For example, some host have addresses assigned by concatenating their Class A network number with the low-order 24 bits of a 48-bit Ethernet hardware address.

<2> Our discussion of Internet broadcasting is based on [6].

<3> If broadcasting is not supported, them presumably a host "knows" the address of a neighbor gateway, and should send the lCMP to that gateway.

<4> This is what was referred to earlier as the coexistence of transparent and explicit subnets on a single network.

References

1. D.R. Boggs, J.F. Shoch, E.A. Taft, and R.M. Metcalfe. "Pup: An Internetwork Architecture." IEEE Transactions on Communications COM-28, 4, pp612-624, April 1980.

2. David D. Clark. Names, Addresses, Ports, and Routes. RFC-814, MIT-LCS, July 1982.

3. Yogan K. Dalal and Robert M. Metcalfe. "Reverse Path Forwarding of Broadcast Packets." Comm. ACM 21, 12, pp1040-1048, December 1978.

4. Ross Finlayson, Timothy Mann, Jeffrey Mogul, Marvin Theimer. A Reverse Address Resolution Protocol. RFC-903, Stanford University, June 1984.

5. R.M. Metcalfe and D.R. Boggs. "Ethernet: Distributed Packet Switching for Local Computer Networks." Comm. ACM 19, 7, pp395-404, July 1976. Also CSL-75-7, Xerox Palo Alto Research Center, reprinted in CSL-80-2.

6. Jeffrey Mogul. Broadcasting Internet Datagrams. RFC-919, Stanford University, October 1984.

7. David Plummer. An Ethernet Address Resolution Protocol. RFC-826, Symbolics, September 1982.

8. Jon Postel. Internet Protocol. RFC-791, USC-ISI, September 1981.

9. Jon Postel. Internet Control Message Protocol. RFC-792, USC-ISI, September 1981.

RFC-919 Broadcasting Internet Datagrams

Jeffrey Mogul
Computer Science Department; Stanford University
October 1974

Status of this Memo

We describe simple rules for broadcasting Internet datagrams on local networks that support broadcast, for addressing broadcasts, and for how gateways should handle them.

This RFC describes a **required** protocol of the Internet Community as of January 1991. Distribution of this memo is unlimited. RFC-922 is a superset of this RFC.

Acknowledgement

This proposal is the result of discussion with several other people, especially J. Noel Chiappa and Christopher A. Kent, both of whom both pointed me at important references.

RFC-919 Broadcasting Internet Datagrams

Introduction

The use of broadcasts, especially on high-speed local area networks, is a good base for many applications. Since broadcasting is not covered in the basic IP specification [RFC-791], there is no agreed-upon way to do it, and so protocol designers have not made use of it. (The issue has been touched upon before, e.g. [IEN-212], but has not been the subject of a standard.)

We consider here only the case of unreliable, unsequenced, possibly duplicated datagram broadcasts (for a discussion of TCP broadcasting, see [IEN-10].) Even though unreliable and limited in length, datagram broadcasts are quite useful [1].

We assume that the data link layer of the local network supports efficient broadcasting. Most common local area networks do support broadcast; for example, Ethernet [7, 5], ChaosNet [10], token ring networks [2], etc.

We do not assume, however, that broadcasts are reliably delivered. (One might consider providing a reliable broadcast protocol as a layer above IP.) It is quite expensive to guarantee delivery of broadcasts; instead, what we assume is that a host will receive most of the broadcasts that are sent. This is important to avoid excessive use of broadcasts; since every host on the network devotes at least some effort to every broadcast, they are costly.

When a datagram is broadcast, it imposes a cost on every host that hears it. Therefore, broadcasting should not be used indiscriminately, but rather only when it is the best solution to a problem.

Note: some organizations have divided their IP networks into subnets, for which a standard [RFC-917] has been proposed. This RFC does not cover the numerous complications arising from the interactions between subnets and broadcasting; see [RFC-922] for a complete discussion.

Terminology

Because broadcasting depends on the specific data link layer in use on a local network, we must discuss it with reference to both physical networks and logical networks.

The terms we will use in referring to physical networks are, from the point of view of the host sending or forwarding a broadcast:

Local Hardware Network

The physical link to which the host is attached.

Remote Hardware Network

A physical network which is separated from the host by at least one gateway.

Collection of Hardware Networks

A set of hardware networks (transitively) connected by gateways.

The IP world includes several kinds of logical network. To avoid ambiguity, we will use the following terms:

Internet

The DARPA Internet collection of IP networks.

IP Network

One or a collection of several hardware networks that have one specific IP network number.

RFC-919 Broadcasting Internet Datagrams

Why Broadcast?

Broadcasts are useful when a host needs to find information without knowing exactly what other host can supply it, or when a host wants to provide information to a large set of hosts in a timely manner.

When a host needs information that one or more of its neighbors might have, it could have a list of neighbors to ask, or it could poll all of its possible neighbors until one responds. Use of a wired-in list creates obvious network management problems (early binding is inflexible). On the other hand, asking all of one's neighbors is slow if one must generate plausible host addresses, and try them until one works. On the ARPANET, for example, there are roughly 65 thousand plausible host numbers. Most IP implementations have used wired-in lists (for example, addresses of "Prime" gateways.) Fortunately, broadcasting provides a fast and simple way for a host to reach all of its neighbors.

A host might also use a broadcast to provide all of its neighbors with some information; for example, a gateway might announce its presence to other gateways.

One way to view broadcasting is as an imperfect substitute for multicasting, the sending of messages to a subset of the hosts on a network. In practice, broadcasts are usually used where multicasts are what is wanted; packets are broadcast at the hardware level, but filtering software in the receiving hosts gives the effect of multicasting.

For more examples of broadcast applications, see [1, 3].

RFC-919 Broadcasting Internet Datagrams

Broadcast Classes

There are several classes of IP broadcasting:

- Single-destination datagram broadcast on the local IP net: A datagrams is destined for a specific IP host, but the sending host broadcasts it at the data link layer, perhaps to avoid having to do routing. Since this is not an IP broadcast, the IP layer is not involved, except that a host should discard datagrams not meant for it without becoming flustered (i.e., printing an error message).
- Broadcast to all hosts on the local IP net: A distinguished value for the host-number part of the IP address denotes broadcast instead of a specific host. The receiving IP layer must be able to recognize this address as well as its own.

  However, it might still be useful to distinguish at higher levels between broadcasts and non-broadcasts, especially in gateways. This is the most useful case of broadcast; it allows a host to discover gateways without wired-in tables, it is the basis for address resolution protocols, and it is also useful for accessing such utilities as name servers, time servers, etc., without requiring wired-in addresses.
- Broadcast to all hosts on a remote IP network: It is occasionally useful to send a broadcast to all hosts on a non-local network; for example, to find the latest version of a hostname database, to bootload a host on an IP network without a bootserver, or to monitor the timeservers on the IP network. This case is the same as local-network broadcasts; the datagram is routed by normal mechanisms until it reaches a gateway attached to the destination IP network, at which point it is broadcast. This class of broadcasting is also known as "directed broadcasting", or quaintly as sending a "letter bomb" [1].
- Broadcast to the entire Internet: This is probably not useful, and almost certainly not desirable.

For reasons of performance or security, a gateway may choose not to forward broadcasts; especially, it may be a good idea to ban broadcasts into or out of an autonomous group of networks.

Broadcast Methods

A host's IP receiving layer must be modified to support broadcasting. In the absence of broadcasting, a host determines if it is the recipient of a datagram by matching the destination address against all of its IP addresses. With broadcasting, a host must compare the destination address not only against the host's addresses, but also against the possible broadcast addresses for that host.

The problem of how best to send a broadcast has been extensively discussed [1, 3, 4, 14, 15]. Since we assume that the problem has already been solved at the data link layer, an IP host wishing to send either a local broadcast or a directed broadcast need only specify the appropriate destination address and send the datagram as usual. Any sophisticated algorithms need only reside in gateways.

RFC-919 Broadcasting Internet Datagrams

Gateways and Broadcasts

Most of the complexity in supporting broadcasts lies in gateways. If a gateway receives a directed broadcast for a network to which it is not connected, it simply forwards it using the usual mechanism. Otherwise, it must do some additional work.

When a gateway receives a local broadcast datagram, there are several things it might have to do with it. The situation is unambiguous, but without due care it is possible to create infinite loops.

The appropriate action to take on receipt of a broadcast datagram depends on several things: the subnet it was received on, the destination network, and the addresses of the gateway.

- The primary rule for avoiding loops is "never broadcast a datagram on the hardware network it was received on". It is not sufficient simply to avoid repeating datagrams that a gateway has heard from itself; this still allows loops if there are several gateways on a hardware network.
- If the datagram is received on the hardware network to which it is addressed, then it should not be forwarded. However, the gateway should consider itself to be a destination of the datagram (for example, it might be a routing table update.)
- Otherwise, if the datagram is addressed to a hardware network to which the gateway is connected, it should be sent as a (data link layer) broadcast on that network. Again, the gateway should consider itself a destination of the datagram.
- Otherwise, the gateway should use its normal routing procedure to choose a subsequent gateway, and send the datagram along to it.

RFC-919 Broadcasting Internet Datagrams

Broadcast IP Addressing - Standards

If different IP implementations are to be compatible, there must be a distinguished number to denote "all hosts".

Since the local network layer can always map an IP address into data link layer address, the choice of an IP "broadcast host number" is somewhat arbitrary. For simplicity, it should be one not likely to be assigned to a real host. The number whose bits are all ones has this property; this assignment was first proposed in [IEN-212]. In the few cases where a host has been assigned an address with a host-number part of all ones, it does not seem onerous to require renumbering.

The address 255.255.255.255 denotes a broadcast on a local hardware network, which must not be forwarded. This address may be used, for example, by hosts that do not know their network number and are asking some server for it.

Thus, a host on net 36, for example, may:

- broadcast to all of its immediate neighbors by using 255.255.255.255
- broadcast to all of net 36 by using 36.255.255.255

(Note that unless the network has been broken up into subnets, these two methods have identical effects.)

If the use of "all ones" in a field of an IP address means "broadcast", using "all zeros" could be viewed as meaning "unspecified". There is probably no reason for such addresses to appear anywhere but as the source address of an ICMP Information Request datagram. However, as a notational convention, we refer to networks (as opposed to hosts) by using addresses with zero fields. For example, 36.0.0.0 means "network number 36" while 36.255.255.255 means "all hosts on network number 36".

RFC-919 Broadcasting Internet Datagrams - Standards

ARP Servers and Broadcasts

The Address Resolution Protocol (ARP) described in [RFC-826] can, if incorrectly implemented, cause problems when broadcasts are used on a network where not all hosts share an understanding of what a broadcast address is. The temptation exists to modify the ARP server so that it provides the mapping between an IP broadcast address and the hardware broadcast address.

This temptation must be resisted. An ARP server should never respond to a request whose target is a broadcast address. Such a request can only come from a host that does not recognize the broadcast address as such, and so honoring it would almost certainly lead to a forwarding loop. If there are N such hosts on the physical network that do not recognize this address as a broadcast, then a datagram sent with a Time-To-Live of T could potentially give rise to T**N spurious re-broadcasts.

8. References

1. David Reeves Boggs. Internet Broadcasting. Ph.D. Th., Stanford University, January 1982.
2. D.D. Clark, K.T. Pogran, and D.P. Reed. "An Introduction to Local Area Networks". Proc. IEEE 66, 11, pp1497-1516, 1978.
3. Yogan Kantilal Dalal. Broadcast Protocols in Packet Switched Computer Networks. Ph.D. Th., Stanford University, April 1977.
4. Yogan K. Dalal and Robert M. Metcalfe. "Reverse Path Forwarding of Broadcast Packets". Comm. ACM 21, 12, pp1040-1048, December 1978.
5. The Ethernet, A Local Area Network: Data Link Layer and Physical Layer Specifications. Version 1.0, Digital Equipment Corporation, Intel, Xerox, September 1980.
6. Robert Gurwitz and Robert Hinden. IP - Local Area Network Addressing Issues. IEN-212, Bolt Beranek and Newman, September 1982.
7. R.M. Metcalfe and D.R. Boggs. "Ethernet: Distributed Packet Switching for Local Computer Networks". Comm. ACM 19, 7, pp395-404, July 1976. Also CSL-75-7, Xerox Palo Alto Research Center, reprinted in CSL-80-2.
8. Jeffrey Mogul. Internet Subnets. RFC-917, Stanford University, October 1984.
9. Jeffrey Mogul. Broadcasting Internet Packets in the Presence of Subnets. RFC-922, Stanford University, October 1984.
10. David A. Moon. Chaosnet. A.I. Memo 628, Massachusetts Institute of Technology Artificial Intelligence Laboratory, June 1981.
11. William W. Plummer. Internet Broadcast Protocols. IEN-10, Bolt Beranek and Newman, March 1977.
12. David Plummer. An Ethernet Address Resolution Protocol. RFC-826, Symbolics, September 1982.
13. Jon Postel. Internet Protocol. RFC 791, ISI, September 1981.
14. David W. Wall. Mechanisms for Broadcast and Selective Broadcast. Ph.D. Th., Stanford University, June 1980.
15. David W. Wall and Susan S. Owicki. Center-based Broadcasting. Computer Systems Lab Technical Report TR189, Stanford University, June 1980.

RFC-922 Broadcasting Internet Datagrams in the Presence of Subnets

Jeffrey Mogul
Computer Science Department; Stanford University
October 1984

Status of this Memo

We specify simple rules for broadcasting Internet datagrams on local networks that support broadcast, for addressing broadcasts, and for how gateways should handle them.

This RFC describes a standard of the Internet Community. As of April 1991 the IAB lists this protocol as **required**. Distribution of this memo is unlimited. RFC-919 presents nearly identical material without mention of subnets. All references to RFC-919 can be resolved by referring to the material contained in this RFC.

Acknowledgement

This proposal here is the result of discussion with several other people, especially J. Noel Chiappa and Christopher A. Kent, both of whom both pointed me at important references.

RFC-922 Broadcasting IP Datagrams in the Presence of Subnets

Introduction

The use of broadcasts, especially on high-speed local area networks, is a good base for many applications. Since broadcasting is not covered in the basic IP specification [RFC-791], there is no agreed-upon way to do it, and so protocol designers have not made use of it. (The issue has been touched upon before, e.g. [IEN-212], but has not been the subject of a standard.)

We consider here only the case of unreliable, unsequenced, possibly duplicated datagram broadcasts (for a discussion of TCP broadcasting, see [IEN-10].) Even though unreliable and limited in length, datagram broadcasts are quite useful [1].

We assume that the data link layer of the local network supports efficient broadcasting. Most common local area networks do support broadcast; for example, Ethernet [7, 5], ChaosNet [10], token ring networks [2], etc.

We do not assume, however, that broadcasts are reliably delivered. (One might consider providing a reliable broadcast protocol as a layer above IP.) It is quite expensive to guarantee delivery of broadcasts; instead, what we assume is that a host will receive most of the broadcasts that are sent. This is important to avoid excessive use of broadcasts; since every host on the network devotes at least some effort to every broadcast, they are costly.

When a datagram is broadcast, it imposes a cost on every host that hears it. Therefore, broadcasting should not be used indiscriminately, but rather only when it is the best solution to a problem.

Note: some organizations have divided their IP networks into subnets, for which a standard [RFC-917] has been proposed. This RFC does not cover the numerous complications arising from the interactions between subnets and broadcasting; see [RFC-922] for a complete discussion.

RFC-922 Broadcasting IP Datagrams in the Presence of Subnets

Terminology

Because broadcasting depends on the specific data link layer in use on a local network, we must discuss it with reference to both physical networks and logical networks.

The terms we will use in referring to physical networks are, from the point of view of the host sending or forwarding a broadcast:

Local Hardware Network

The physical link to which the host is attached.

Remote Hardware Network

A physical network which is separated from the host by at least one gateway.

Collection of Hardware Networks

A set of hardware networks (transitively) connected by gateways.

The IP world includes several kinds of logical network. To avoid ambiguity, we will use the following terms:

Internet

The DARPA Internet collection of IP networks.

IP Network

One or a collection of several hardware networks that have one specific IP network number.

Subnet

A single member of the collection of hardware networks that compose an IP network. Host addresses on a given subnet share an IP network number with hosts on all other subnets of that IP network, but the local-address part is divided into subnet-number and host-number fields to indicate which subnet a host is on. We do not assume a particular division of the local-address part; this could vary from network to network.

The introduction of a subnet level in the addressing hierarchy is at variance with the IP specification [12], but as the use of addressable subnets proliferates it is obvious that a broadcasting scheme should support subnetting. For more on subnets, see [8].

In this paper, the term "host address" refers to the host-on-subnet address field of a subnetted IP network, or the host-part field otherwise.

An IP network may consist of a single hardware network or a collection of subnets; from the point of view of a host on another IP network, it should not matter.

RFC-922 Broadcasting IP Datagrams in the Presence of Subnets

Why Broadcast?

Broadcasts are useful when a host needs to find information without knowing exactly what other host can supply it, or when a host wants to provide information to a large set of hosts in a timely manner.

When a host needs information that one or more of its neighbors might have, it could have a list of neighbors to ask, or it could poll all of its possible neighbors until one responds. Use of a wired-in list creates obvious network management problems (early binding is inflexible). On the other hand, asking all of one's neighbors is slow if one must generate plausible host addresses, and try them until one works. On the ARPANET, for example, there are roughly 65 thousand plausible host numbers. Most IP implementations have used wired-in lists (for example, addresses of "Prime" gateways.) Fortunately, broadcasting provides a fast and simple way for a host to reach all of its neighbors.

A host might also use a broadcast to provide all of its neighbors with some information; for example, a gateway might announce its presence to other gateways.

One way to view broadcasting is as an imperfect substitute for multicasting, the sending of messages to a subset of the hosts on a network. In practice, broadcasts are usually used where multicasts are what is wanted; datagrams are broadcast at the hardware level, but filtering software in the receiving hosts gives the effect of multicasting.

For more examples of broadcast applications, see [1, 3].

RFC-922 Broadcasting IP Datagrams in the Presence of Subnets

Broadcast Classes

There are several classes of IP broadcasting:

- Single-destination datagram broadcast on the local IP net: A datagrams is destined for a specific IP host, but the sending host broadcasts it at the data link layer, perhaps to avoid having to do routing. Since this is not an IP broadcast, the IP layer is not involved, except that a host should discard datagrams not meant for it without becoming flustered (i.e., printing an error message).
- Broadcast to all hosts on the local IP net: A distinguished value for the host-number part of the IP address denotes broadcast instead of a specific host. The receiving IP layer must be able to recognize this address as well as its own.

  However, it might still be useful to distinguish at higher levels between broadcasts and non-broadcasts, especially in gateways. This is the most useful case of broadcast; it allows a host to discover gateways without wired-in tables, it is the basis for address resolution protocols, and it is also useful for accessing such utilities as name servers, time servers, etc., without requiring wired-in addresses.
- Broadcast to all hosts on a remote IP network: It is occasionally useful to send a broadcast to all hosts on a non-local network; for example, to find the latest version of a hostname database, to bootload a host on an IP network without a bootserver, or to monitor the timeservers on the IP network. This case is the same as local-network broadcasts; the datagram is routed by normal mechanisms until it reaches a gateway attached to the destination IP network, at which point it is broadcast. This class of broadcasting is also known as "directed broadcasting", or quaintly as sending a "letter bomb" [1].
- Broadcast to the entire Internet: This is probably not useful, and almost certainly not desirable.

For reasons of performance or security, a gateway may choose not to forward broadcasts; especially, it may be a good idea to ban broadcasts into or out of an autonomous group of networks.

RFC-922 Broadcasting IP Datagrams in the Presence of Subnets

Broadcast Methods

A host's IP receiving layer must be modified to support broadcasting. In the absence of broadcasting, a host determines if it is the recipient of a datagram by matching the destination address against all of its IP addresses. With broadcasting, a host must compare the destination address not only against the host's addresses, but also against the possible broadcast addresses for that host.

The problem of how best to send a broadcast has been extensively discussed [1, 3, 4, 13, 14]. Since we assume that the problem has already been solved at the data link layer, an IP host wishing to send either a local broadcast or a directed broadcast need only specify the appropriate destination address and send the datagram as usual. Any sophisticated algorithms need only reside in gateways.

The problem of broadcasting to all hosts on a subnetted IP network is apparently somewhat harder. However, even in this case it turns out that the best known algorithms require no additional complexity in non-gateway hosts. A good broadcast method will meet these additional criteria:

- No modification of the IP datagram format.
- Reasonable efficiency in terms of the number of excess copies generated and the cost of paths chosen.
- Minimization of gateway modification, in both code and data space.
- High likelihood of delivery.

The algorithm that appears best is the Reverse Path Forwarding (RPF) method [4]. While RPF is suboptimal in cost and reliability, it is quite good, and is extremely simple to implement, requiring no additional data space in a gateway.

RFC-922 Broadcasting IP Datagrams in the Presence of Subnets

Gateways and Broadcasts

Most of the complexity in supporting broadcasts lies in gateways. If a gateway receives a directed broadcast for a network to which it is not connected, it simply forwards it using the usual mechanism. Otherwise, it must do some additional work.

RFC-922 Broadcasting IP Datagrams in the Presence of Subnets - Gateways

Local Broadcasts

When a gateway receives a local broadcast datagram, there are several things it might have to do with it. The situation is unambiguous, but without due care it is possible to create infinite loops.

The appropriate action to take on receipt of a broadcast datagram depends on several things: the subnet it was received on, the destination network, and the addresses of the gateway.

- The primary rule for avoiding loops is "never broadcast a datagram on the hardware network it was received on". It is not sufficient simply to avoid repeating datagram that a gateway has heard from itself; this still allows loops if there are several gateways on a hardware network.
- If the datagram is received on the hardware network to which it is addressed, then it should not be forwarded. However, the gateway should consider itself to be a destination of the datagram (for example, it might be a routing table update.)
- Otherwise, if the datagram is addressed to a hardware network to which the gateway is connected, it should be sent as a (data link layer) broadcast on that network. Again, the gateway should consider itself a destination of the datagram.
- Otherwise, the gateway should use its normal routing procedure to choose a subsequent gateway, and send the datagram along to it.

Multi-subnet broadcasts

When a gateway receives a broadcast meant for all subnets of an IP network, it must use the Reverse Path Forwarding algorithm to decide what to do. The method is simple: the gateway should forward copies of the datagram along all connected links, if and only if the datagram arrived on the link which is part of the best route between the gateway and the source of the datagram. Otherwise, the datagram should be discarded.

This algorithm may be improved if some or all of the gateways exchange among themselves additional information; this can be done transparently from the point of view of other hosts and even other gateways. See [4, 3] for details.

Pseudo-Algol Routing Algorithm

This is a pseudo-Algol description of the routing algorithm a gateway should use. The algorithm is shown in figure 1. Some definitions are:

RouteLink(host)

A function taking a host address as a parameter and returning the first-hop link from the gateway to the host.

RouteHost(host)

As above but returns the first-hop host address.

ResolveAddress(host)

Returns the hardware address for an IP host.

IncomingLink

The link on which the packet arrived.

OutgoingLinkSet

The set of links on which the packet should be sent.

OutgoingHardwareHost

The hardware host address to send the packet to.

Destination.host

The host-part of the destination address.

Destination.subnet

The subnet-part of the destination address.

Destination.ipnet

The IP-network-part of the destination address.

```
BEGIN
    IF Destination.ipnet IN AllLinks THEN
        BEGIN
            IF IsSubnetted(Destination.ipnet) THEN
                BEGIN
                    IF Destination.subnet = BroadcastSubnet THEN
                        BEGIN        /* use Reverse Path Forwarding algorithm */
                            IF IncomingLink = RouteLink(Source) THEN
                                BEGIN IF Destination.host = BroadcastHost THEN
                                    OutgoingLinkSet <- AllLinks -
                                    IncomingLink;
                                    OutgoingHost <- BroadcastHost;
                                    Examine packet for possible internal use;
                                END
                            ELSE   /* duplicate from another gateway, discard */
                                Discard;
                        END
```

```
                ELSE
                    IF Destination.subnet = IncomingLink.subnet THEN
                        BEGIN              /* forwarding would cause a loop */
                            IF Destination.host = BroadcastHost THEN
                                Examine packet for possible internal use;
                                Discard;
                        END
                    ELSE BEGIN      /* forward to (possibly local) subnet */
                        OutgoingLinkSet <- RouteLink(Destination);
                        OutgoingHost <- RouteHost(Destination);
                    END
            END
        ELSE BEGIN            /* destined for one of our local networks */
            IF Destination.ipnet = IncomingLink.ipnet THEN
                BEGIN                  /* forwarding would cause a loop */
                    IF Destination.host = BroadcastHost THEN
                        Examine packet for possible internal use;
                        Discard;
                END
            ELSE BEGIN                          /* might be a broadcast */
                OutgoingLinkSet <- RouteLink(Destination);
                OutgoingHost <- RouteHost(Destination);
                END
        END
    END
ELSE BEGIN                        /* forward to a non-local IP network */
    OutgoingLinkSet <- RouteLink(Destination);
    OutgoingHost <- RouteHost(Destination);
    END
OutgoingHardwareHost <- ResolveAddress(OutgoingHost);
END
```

Figure 1: Pseudo-Algol algorithm for routing broadcasts by gateways

RFC-922 Broadcasting IP Datagrams in the Presence of Subnets

Broadcast IP Addressing - Conventions

If different IP implementations are to be compatible, there must be convention distinguished number to denote "all hosts" and "all subnets".

Since the local network layer can always map an IP address into data link layer address, the choice of an IP "broadcast host number" is somewhat arbitrary. For simplicity, it should be one not likely to be assigned to a real host. The number whose bits are all ones has this property; this assignment was first proposed in [6]. In the few cases where a host has been assigned an address with a host-number part of all ones, it does not seem onerous to require renumbering.

The "all subnets" number is also all ones; this means that a host wishing to broadcast to all hosts on a remote IP network need not know how the destination address is divided up into subnet and host fields, or if it is even divided at all. For example, 36.255.255.255 may denote all the hosts on a single hardware network, or all the hosts on a subnetted IP network with 1 byte of subnet field and 2 bytes of host field, or any other possible division.

The address 255.255.255.255 denotes a broadcast on a local hardware network that must not be forwarded. This address may be used, for example, by hosts that do not know their network number and are asking some server for it.

Thus, a host on net 36, for example, may:

- broadcast to all of its immediate neighbors by using 255.255.255.255
- broadcast to all of net 36 by using 36.255.255.255

without knowing if the net is subnetted; if it is not, then both addresses have the same effect. A robust application might try the former address, and if no response is received, then try the latter. See [1] for a discussion of such "expanding ring search" techniques.

If the use of "all ones" in a field of an IP address means "broadcast", using "all zeros" could be viewed as meaning "unspecified". There is probably no reason for such addresses to appear anywhere but as the source address of an ICMP Information Request datagram. However, as a notational convention, we refer to networks (as opposed to hosts) by using addresses with zero fields. For example, 36.0.0.0 means "network number 36" while 36.255.255.255 means "all hosts on network number 36".

RFC-922 Broadcasting IP Datagrams in the Presence of Subnets - Conventions

ARP Servers and Broadcasts

The Address Resolution Protocol (ARP) described in [RFC_826] can, if incorrectly implemented, cause problems when broadcasts are used on a network where not all hosts share an understanding of what a broadcast address is. The temptation exists to modify the ARP server so that it provides the mapping between an IP broadcast address and the hardware broadcast address.

This temptation must be resisted. An ARP server should never respond to a request whose target is a broadcast address. Such a request can only come from a host that does not recognize the broadcast address as such, and so honoring it would almost certainly lead to a forwarding loop. If there are N such hosts on the physical network that do not recognize this address as a broadcast, then a datagram sent with a Time-To-Live of T could potentially give rise to T**N spurious re-broadcasts.

8. References

1. David Reeves Boggs. Internet Broadcasting. Ph.D. Th., Stanford University, January 1982.
2. D.D. Clark, K.T. Pogran, and D.P. Reed. "An Introduction to Local Area Networks". Proc. IEEE 66, 11, pp1497-1516, November 1978.
3. Yogan Kantilal Dalal. Broadcast Protocols in Packet Switched Computer Networks. Ph.D. Th., Stanford University, April 1977.
4. Yogan K. Dalal and Robert M. Metcalfe. "Reverse Path Forwarding of Broadcast Packets". Comm. ACM 21, 12, pp1040-1048, December 1978.
5. The Ethernet, A Local Area Network: Data Link Layer and Physical Layer Specifications. Version 1.0, Digital Equipment Corporation, Intel, Xerox, September 1980.
6. Robert Gurwitz and Robert Hinden. IP - Local Area Network Addressing Issues. IEN-212, BBN, September 1982.
7. R.M. Metcalfe and D.R. Boggs. "Ethernet: Distributed Packet Switching for Local Computer Networks". Comm. ACM 19, 7, pp395-404, July 1976. Also CSL-75-7, Xerox Palo Alto Research Center, reprinted in CSL-80-2.
8. Jeffrey Mogul. Internet Subnets. RFC-917, Stanford University, October 1984.
9. David A. Moon. Chaosnet. A.I. Memo 628, Massachusetts Institute of Technology Artificial Intelligence Laboratory, June 1981.
10. William W. Plummer. Internet Broadcast Protocols. IEN-10, BBN, March 1977.
11. David Plummer. An Ethernet Address Resolution Protocol. RFC-826, Symbolics, September 1982.
12. Jon Postel. Internet Protocol. RFC-791, ISI, September 1981.
13. David W. Wall. Mechanisms for Broadcast and Selective Broadcast. Ph.D. Th., Stanford University, June 1980.
14. David W. Wall and Susan S. Owicki. Center-based Broadcasting. Computer Systems Lab Technical Report TR189, Stanford University, June 1980.

Output Marking Telnet Option

S. Silverman MITRE - Washington
January 1985

Status of this Memo

This RFC proposes a new option for Telnet for the ARPA-Internet community, and requests discussion and suggestions for improvements. Distribution of this memo is unlimited.

Overview

This proposed option would allow a Server-Telnet to send a banner to a User-Telnet so that this banner would be displayed on the workstation screen independently of the application software running in the Server-Telnet.

Command Name and Code

OUTMRK 27

Default

WON'T OUTMRK
DON'T OUTMRK

Output marking information will not be exchanged.

Command Meanings

IAC WILL OUTMRK

Sender is willing to send output marking information in a subsequent sub-negotiation.

IAC WON'T OUTMRK

Sender refuses to send output marking information.

IAC DO OUTMRK

Sender is willing to receive output marking information in a subsequent sub-negotiation.

IAC DON'T OUTMRK

Sender refuses to accept output marking information.

IAC SB OUTMRK CNTL data IAC SE

The sender requests receiver to use the data in this subnegotiation as a marking for the normally transmitted Telnet data until further notice. The CNTL octet indicates the position of the marking.

IAC SB OUTMRK ACK IAC SE

The sender acknowledges the data and agrees to use it to perform output marking.

IAC SB OUTMRK NAK IAC SE

The sender objects to using the data to perform output marking.

RFC-933 Output Marking Telnet Option

Motivation for the Option

The security architecture of some military systems identifies a security level with each Telnet connection. There is a corresponding need to display a security banner on visual display devices. (Reference: Department of Defense Trusted Computer System Evaluation Criteria, Section 3.1.1.3.2.3, Labeling Human-Readable Output.)

The output marking is currently done by transmitting the banner as data within each screen of data. It would be more efficient to transmit the data once with instructions and have User-Telnet maintain the banner automatically without any additional Server-Telnet action. This frees Server-Telnet from needing to know the output device page size.

Under this proposal Server-Telnet would send an option sequence with the command, a control flag, and the banner to be used. While current systems use the top of the screen, it is conceivable other systems would want to put the banner at the bottom or perhaps even the side of the screen. This is the reason for the control flag.

RFC-933 Output Marking Telnet Option

Description of the Option

Either side of the session can initiate the option; however, normally it will be the server side that initiates the request to perform output marking. Either the Server-Telnet sends "WILL OUTMRK" or the User-Telnet sends a "DO OUTMRK". The party receiving the initial "WILL" (or "DO") would respond with "DO" (or "WILL") to accept the option. Then Server-Telnet responds with the marking data. The format of this is:

"IAC SB OUTMRK CNTL data IAC SE"

CNTL is the Control Flag described below, the data is in ASCII.

If this is satisfactory, User-Telnet responds:

"IAC SB OUTMRK ACK IAC SE"

ACK is the ASCII ACK (6).

From this point, User-Telnet will have to translate any command which uses cursor controls so that the application data is mapped to the application part of the screen.

If the data passed in the subnegotiation field is unacceptable to User-Telnet, then it responds with:

"IAC SB OUTMRK NAK IAC SE"

NAK is the ASCII NAK (21).

It is now up to Server-Telnet to start the sequence over again and use "more acceptable" data (or possibly take other action such as connection termination).

To terminate output marking, Server-Telnet transmits "WON'T OUTMRK".

If necessary, User-Telnet would notify Server-Telnet about the new effective page size. User-Telnet would then map the output data to the allowed usable space on the screen.

User-Telnet may request OUTMRK data or initiate setup of this convention at anytime by transmitting "DO OUTMRK". If a WILL, DO OUTMRK exchange is not followed by the OUTMRK subnegotiation of the marking data, the User-Telnet may terminate the output marking option by sending a "DON'T OUTMRK".

Control Flag

The CNTL flag is defined as:

D = Default, the placement of the markings is up to User-Telnet. This is the expected mode for most interactions.

T = Top, this banner is to be used as the top of the screen. If multiple output markings are desired, then T and B (or R & L) are to be used.

B = Bottom, this banner is to be used at the bottom of the screen.

L = Left, markings on the left. (The precise meaning of this is to be defined.)

R = Right, marking on right. (The precise meaning of this is to be defined.)

Banner Data

The use of Carriage Return and Line Feed (CRLF) will be interpreted as a end of line in the marking banner text. If the user wants a multiline banner, CRLF will be used between each line. No CRLF is needed at the end of the marking data.

To use multiple banners, all of the banners will be included in one subnegotiation command

of the form:

"IAC SB OUTMRK CNTL data GS CNTL data IAC SE"

where GS is the ASCII Group Separator (29) character.

User-Telnet will be responsible for positioning the marking banner data on the screen.

Post Office Protocol - Version 2

M. Butler, J. Postel, D. Chase, J. Goldberger, J. K. Reynolds
USC/Information Sciences Institute
February 1985

Status of this Memo

This RFC suggests a simple method for workstations to dynamically access mail from a mailbox server. This RFC specifies a proposed protocol for the ARPA-Internet community, and requests discussion and suggestions for improvement. This memo is a revision of RFC 918. Distribution of this memo is unlimited.

RFC-937 Post Office Protocol - Version 2

Introduction

The intent of the Post Office Protocol Version 2 (POP2) is to allow a user's workstation to access mail from a mailbox server. It is expected that mail will be posted from the workstation to the mailbox server via the Simple Mail Transfer Protocol (SMTP). For further information see RFC-821 [RFC-821] and RFC-822 [RFC-822].

This protocol assumes a reliable data stream such as provided by TCP or any similar protocol. When TCP is used, the POP2 server listens on port 109 [RFC-1060].

System Model and Philosophy

While we view the workstation as an Internet host in the sense that it implements IP, we do not expect the workstation to contain the user's mailbox. We expect the mailbox to be on a server machine.

We believe it is important for the mailbox to be on an "always up" machine and that a workstation may be frequently powered down, or otherwise unavailable as an SMTP server.

POP2 is designed for an environment of workstations and servers on a low-delay, high-throughput, local networks (such as Ethernets). POP2 may be useful in other environments as well, but if the environment is substantially different, a different division of labor between the client and server may be appropriate, and a different protocol required.

Suppose the user's real name is John Smith, the user's machine is called FIDO, and that the mailbox server is called DOG-HOUSE. Then we expect the user's mail to be addressed to JSmith@DOG-HOUSE.ARPA (not JSmith@FIDO.ARPA).

That is, the destination of the mail is the mailbox on the server machine. The POP2 protocol and the workstation are merely a mechanism for viewing the messages in the mailbox.

The user is not tied to any particular workstation for accessing his mail. The workstation does not appear as any part of the mailbox address.

This is a very simple protocol. This is not a user interface. We expect that there is a program in the workstation that is friendly to the user. This protocol is not "user friendly". One basic rule of this protocol is "if anything goes wrong close the connection". Another basic rule is to have few options.

POP2 does not parse messages in any way. It does not analyze message headers (Date:, From:, To:, Cc:, or Subject:). POP2 simply transmits whole messages from a mailbox server to a client workstation.

The Protocol

The POP2 protocol is a sequence of commands and replies. The design draws from many previous protocols of the ARPA-Internet community.

The server must be listening for a connection. When a connection is opened the server sends a greeting message and waits for commands. When commands are received the server acts on them and responds with replies.

The client opens a connection, waits for the greeting, then sends the HELO command with the user name and password arguments to establish authorization to access mailboxes. The server returns the number of messages in the default mailbox.

The client may read the default mailbox associated with the user name or may select another mailbox by using the FOLD command. The server returns the number of messages in the mailbox selected.

The client begins a message reading transaction with a READ command. The read command may optionally indicate which message number to read, the default is the current message (incremented when a message is read and set to one when a new folder is selected). The server returns the number of characters in the message.

The client asks for the content of the message to be sent with the RETR command. The server sends the message data.

When all the data has been received the client sends an acknowledgment command. This is one of ACKS, ACKD, and NACK.

ACKS means "I've received the message successfully and please keep it in the mailbox".

ACKD means "I've received the message successfully and please delete it from the mailbox".

NACK means "I did not receive the message and please keep it in the mailbox".

In the case of ACKS or ACKD the server increments the current message indicator. In the case of NACK the current message indicator stays the same.

In all cases the server returns the number of characters in the (now) current message.

The client terminates the session with the QUIT command. The server returns an ok.

The Normal Scenario

Client			Server
			Wait for Connection
Open Connection	-->		
		<--	+ POP2 Server Ready
			Wait for Command
HELO Fred Secret	-->		
		<--	#13 messages for you
			Wait for Command
READ 13	-->		
		<--	=537 characters in that message
			Wait for Command
RETR	-->		
		<--	(send the message data)
			Wait for Command
ACKS	-->		
		<--	=0 no more messages
			Wait for Command
QUIT	-->		
		<--	+ OK
Close connection	-->	<--	Close connection
			Wait for Connection (go back to start)

Conventions

Arguments

These arguments have system specific definitions.

user - A login account name.

password - The password for the login account.

mailbox - A mailbox name (also called a mail folder).

Default Mailboxes

TOPS-20

MAIL.TXT.1 - from login directory

UNIX

both

/usr/spool/mail/user

and

/usr/user/Mail/inbox/*

where "user" is the user value supplied in the HELO command.

End of Line

End of Line is Carriage Return (CR) followed by Line Feed (LF). This sequence is indicated by "CRLF" in this document. This end of line convention must be used for commands and replies.

Message Length

The reply to the READ command or an acknowledgment command (ACKS, ACKD, NACK) is the length (a character count) of the next message to be transmitted. This includes all the characters in the data transmitted. CRLF counts as two characters. A length of zero means the message does not exist or is empty. A request to transmit a message of zero length will result in the server closing the connection. The message is transmitted in the standard internet format described in RFC-822 [2] and NVT-ASCII. This may be different from the storage format and may make computing the message length from the stored message non-trivial.

Message Numbers

The reply to the HELO and FOLD commands is a count of the number of messages in a the selected mailbox. The READ command has a message number as an optional argument. These numbers are decimal, start at one, and computed with respect to the current mailbox. That is, the first message in a mailbox is message number 1.

Numbers

All numbers in this memo and protocol are decimal.

Quoting

In a few cases, there may be a need to have a special character in an argument (user, password, or mailbox) that is not allowed by the syntax. For example, a space in a password. To allow for this, a quoting convention is defined. Unfortunately, such quoting conventions "use up" another otherwise uninteresting character. In this protocol the

back slash "\" is used as the quote character. To include a space in an argument the two character sequence "back-slash, space" is transmitted. To include a back-slash in an argument the two character sequence "back-slash, back-slash" is transmitted. This quoting convention is used in the command arguments only, it is not used in the mail data transmitted in response to a RETR command.

Reply Strings

The first character is required to be as specified (i.e., "+", "-", "=", "#"). The optional strings that follow can be whatever the implementer thinks is appropriate.

RFC-937 Post Office Protocol - Version 2

Definitions of Commands and Replies

Commands	Replies
HELO user password	Greeting
FOLD mailbox	+ OK
READ [n]	- Error
RETR	#xxx
ACKS	=yyy
ACKD	
NACK	
QUIT	

HELO user password

The Hello command identifies the user to the server and carries the password authenticating this user. This information is used by the server to control access to the mailboxes. The Hello command is the "HELO" keyword, followed by the user argument, followed by the password argument, followed by CRLF.

Possible responses:

"#nnn"

where nnn is the number of messages in the default mailbox,"

"- error report" and Close the connection.

FOLD mailbox

The Folder command selects another mailbox or mail folder. The server must check that the user is permitted read access to this mailbox. If the mailbox is empty or does not exist, the number of messages reported is zero. The Folder command is the "FOLD" keyword, followed by the mailbox argument, followed by CRLF.

Possible responses:

"#nnn"

where nnn is the number of messages in this mailbox.

READ [nnn]

The Read command begins a message reading transaction. If the Read command is given without an argument the current message is implied (the current message indicator is incremented by the ACKS or ACKD commands). If an argument is used with the Read command it is the message number to be read, and this command sets the current message indicator to that value. The server returns the count of characters in the message to be transmitted. If there is no message to be read, the count of zero is returned. If the message was previously deleted with the ACKD command, the count of zero is returned. The Read command is followed by the RETR command, the READ command, the FOLD command, or the QUIT command. Do not attempt to RETR a message of zero characters. The Read command is the "READ" keyword, optionally followed by the message number argument, followed by CRLF.

Possible responses:

"=ccc"

where ccc is the number of characters in this message.

RETR

The Retrieve command confirms that the client is ready to receive the mail data. It must be followed by an acknowledgment command. The server will close the connection if asked to transmit a message of zero characters (i.e., transmit a non-existent message). The message is transmitted according to the Internet mail format standard RFC-822 [RFC-822] in NVT-ASCII. The Retrieve command is the "RETR" keyword, followed by CRLF.

Possible responses:

the message data

Close the connection

ACKS

The Acknowledge and Save command confirms that the client has received and accepted the message. The ACKS command ends the message reading transaction. The message is kept in the mailbox. The current message indicator is incremented. The server returns the count of characters in the now current message to be transmitted. If there is no message to be read or the message is marked deleted, the count of zero is returned. The Acknowledge and Save command is the "ACKS" keyword, followed by CRLF.

Possible responses:

"=ccc"

where ccc is the number of characters in the next message.

ACKD

The Acknowledge and Delete command confirms that the client has received and accepted the message. The ACKD command ends the message reading transaction. If the user is authorized to have write access to the mailbox, the message is deleted from the mailbox. Actually, the message is only marked for deletion. The actual change is made when the mailbox is released at the end of the session or when the client selects another mailbox with the FOLD command. The messages are not renumbered until the mailbox is released. If the user does not have write access to the mailbox no change is made to the mailbox. The response is the same whether or not the message was actually deleted. The current message indicator is incremented. The server returns the count of characters in the now current message to be transmitted. If there is no message to be read or the message is marked deleted, the count of zero is returned. The Acknowledge and Delete command is the "ACKD" keyword, followed by CRLF.

Possible responses:

"=ccc"

where ccc is the number of characters in the next message.

NACK

The Negative Acknowledge command reports that the client did not receive the message. The NACK command ends the message reading transaction. The message is kept in the mailbox. The current message indicator remains the same. The server returns the count of characters in the current message. Since the count to be returned is for the message just transmitted it the message must exist and not be marked deleted, and the count must be positive (non-zero). The Negative Acknowledge command is the "NACK" keyword, followed by CRLF.

Possible responses:

"=ccc"

where ccc is the number of characters in this message.

QUIT

The Quit command indicates the client is done with the session. The server sends an OK response and then closes the connection. The Quit command is the "QUIT" keyword, followed by CRLF.

Possible responses:

"+ OK" and Close the connection

Greeting

The greeting is sent by the server as soon as the connection is established. The greeting is a plus sign, followed by the protocol name ("POP2"), followed by the server host name, optionally followed by text, and ending with a CRLF.

RFC-937 Post Office Protocol - Version 2: Replies

+ OK

The success or plus sign response indicates successful completion of the operation specified in the command. The success response is a plus sign, optionally followed by text, and ending with a CRLF.

- Error

The failure or minus sign response indicates the failure of the operation specified in the command. The failure response is a minus sign, optionally followed by text, and ending with a CRLF.

RFC-937 Post Office Protocol - Version 2: Replies

#xxx

The count or number sign response tells the number of messages in a folder or mailbox referenced by the command. The count response is a number sign, followed by a number, optionally followed by text, and ending with a CRLF.

=yyy

The length or equal sign response tells the length in characters of the message referenced by the command. The length response is a equal sign, followed by a number, optionally followed by text, and ending with a CRLF.

Timeouts

In any protocol of this type there have to be timeouts. Neither side wants to get stuck waiting forever for the other side (particularly is the other side has gone crazy or crashed).

The client expects a reply to a command fairly quickly and so should have a short timeout for this. This timeout is called T1.

For some servers, it may take some processing to compute the number of messages in a mailbox, or the length of a message, or to reformat a stored message for transmission, so this time out has to allow for such processing time. Also care must be taken not to timeout waiting for the completion of a RETR reply while a long message is in fact being transfered.

The server expects the session to progress with some but not excessive delay between commands and so should have a long timeout waiting for the next command. This time out is T2.

One model of use of this protocol is that any number of different types of clients can be built with different ways of interacting with the human user and the server, but still expecting the client to open the connection to the server, present a sequence of commands, and close the connection, without waiting for intervention by the human user. With such client implementations, it is reasonable for the server to have a fairly small value for timeout T2.

On the other hand, one could easily have the client be very human user directed with the user making decisions between commands. This would cause arbitrary delays between client commands to the server, and require the value of timeout T2 to be quite large.

Implementation Discussion

Comments on a Server on TOPS-20

On TOPS-20, a mailbox is a single file. New messages are appended to the file. There is a separator line between messages.

The tricky part of implementing a POP2 server on TOPS-20 is to provide for deleting messages. This only has to be done for the mailboxes (files) for which the user has write access. The problem is to avoid both (1) preventing other users from accessing or updating the mailbox for long periods, and (2) accidentally deleting a message the user has not seen.

One suggestion is as follows:

When a mailbox is first selected, if the user has write access, rename the mailbox file to some temporary name. Thus new messages will be placed in a new instance of the mailbox file. Conduct all POP2 operation on the temporary mailbox file (including deleting messages). When the POP2 session is over or another mailbox is selected, prepend any messages left undeleted in the temporary file to the new instance of the mailbox file.

Sizes

The maximum length of a command line is 512 characters (including the command word and the CRLF).

The maximum length of a reply line is 512 characters (including the success indicator (+, -, =, #) and the CRLF).

The maximum length of a text line is 1000 characters (including CRLF).

ISI has developed a POP2 server for TOPS-20 and for Berkeley 4.2 Unix, and a POP2 client for an IBM-PC and for Berkeley 4.2 Unix.

Extensions Not Supported

POP2 does not examine the internal data of messages. In particular, the server does not parse message headers.

The server doesn't have any state information (i.e., it doesn't know from one session to the next what has happened). For example, the server doesn't know which messages were received since the last time the user used POP2, so it can't send just the "new" messages.

Examples

Example 1 - Read default mailbox

Client		Server
		Wait for connection
Open connection	-->	
		<-- + POP2 USC-ISIF.ARPA Server
HELO POSTEL SECRET	-->	
		<-- #2 messages in your mailbox
READ	-->	
		<-- =537 characters in message 1
RETR	-->	
		<-- [data of message 1]
ACKD	-->	
		<-- =234 characters in message 2
RETR	-->	
		<-- [data of message 2]
ACKD	-->	
		<-- =0 no more messages
QUIT	-->	
		<-- + OK, bye, bye
Close connection	-->	<-- Close connection
		Go back to start

Example 2:

Client		**Server**
		Wait for connection
Open connection	-->	
		<-- + POP2 ISI-VAXA.ARPA server here
HELO smith secret	-->	
		<-- #35 messages
FOLD /usr/spool/mail/smith	-->	
		<-- #27 messages
READ 27	-->	
		<-- =10123 characters in that message
RETR	-->	
		<-- [data of message 27]
ACKS	-->	
		<-- =0 no more messages
QUIT	-->	
		<-- + bye, call again sometime.
Close connection	-->	<-- Close connection
		Go back to start

Example 3:

Client		**Server**
		Wait for connection
Open connection	-->	
	<--	+ POP2 ISI-VAXA.ARPA server here
HELO Jones secret	-->	
	<--	#0 messages
READ	-->	
	<--	Close connection
Close connection	-->	
		Go back to start

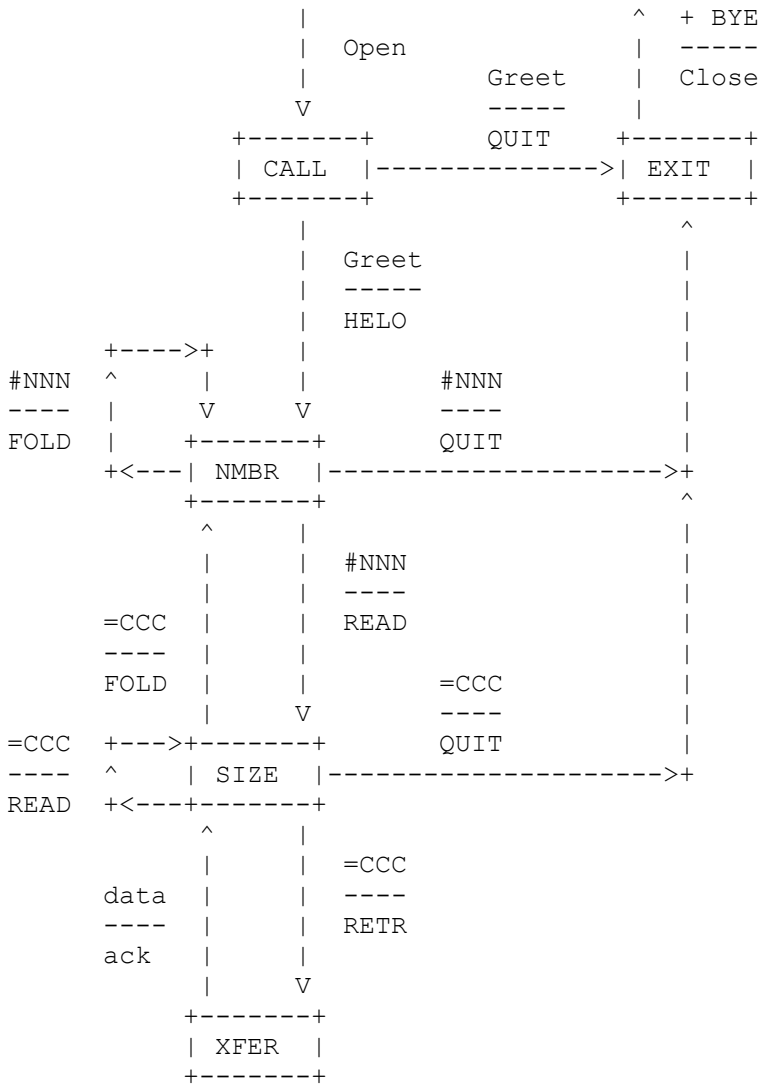
Formal Syntax

```
 <digit>    = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
 <letter>   = A | B | C | ... | Z
              a | b | c | ... | z
 <punct>    = ! | " | # | $ | % | & | ' | ( | ) | * |
              + | , | - | / | : | < | = | > | ? | @ |
              [ | ] | ^ | _ | ` | { | | | } | ~
<quote>     = \
<any>       = any one of the 128 ASCII codes
<CR>        = carriage return, code 10
<LF>        = line feed, code 13
<SP>        = space, code 32
<CRLF>      = <CR> <LF>
<print>     = <letter> | <digit> | <punct> | <quote> <any>
<char>      = <print> | <SP>
<word>      = <print> | <print> <word>
<string>    = <char> | <char> <string>
<ld>        = <letter> | <digit>
<ldh>       = <letter> | <digit> | -
<ldhs>      = <ldh> | <ldh> <ldhs>
<name>      = <letter> [ [ <ldhs> ] <ld> ]
<host>      = <name> | <name> . <host>
<user>      = <word>
<password>
            = <word>
<mailbox>   = <string>
<number>    = <digit> | <digit> <number>
<helo>      = HELO <SP> <user> <SP> <password> <CRLF>
<fold>      = FOLD <SP> <mailbox> <CRLF>
<read>      = READ [<SP> <number>] <CRLF>
<retr>      = RETR <CRLF>
<acks>      = ACKS <CRLF>
<ackd>      = ACKD <CRLF>
<nack>      = NACK <CRLF>
<quit>      = QUIT <CRLF>
<ok>        = + [<SP> <string>] <CRLF>
<err>       = - [<SP> <string>] <CRLF>
<count>     = # <number> [<SP> <string>] <CRLF>
<greet>     = + <SP> POP2 <SP> <host> [<SP> <string>] <CRLF>
<length>    = = <number> [<SP> <string>] <CRLF>
<command>
            = <helo> | <fold> | <read> | <retr> |
              <acks> | <ackd> | <nack> | <quit>
<reply>     = <ok> | <err> | <count> | <length> | <greet>
```

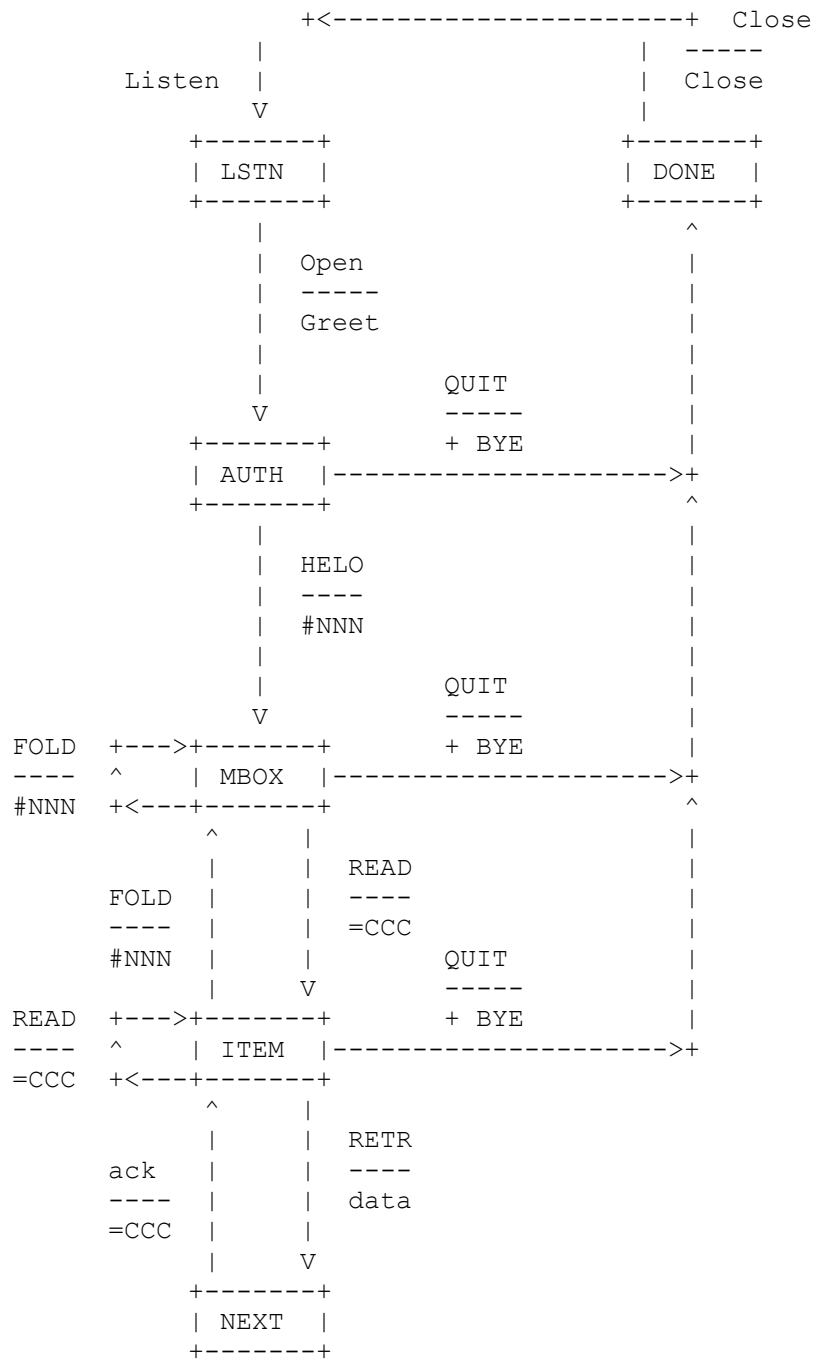
State Diagrams

RFC-937 Post Office Protocol - Version 2: State Diagrams

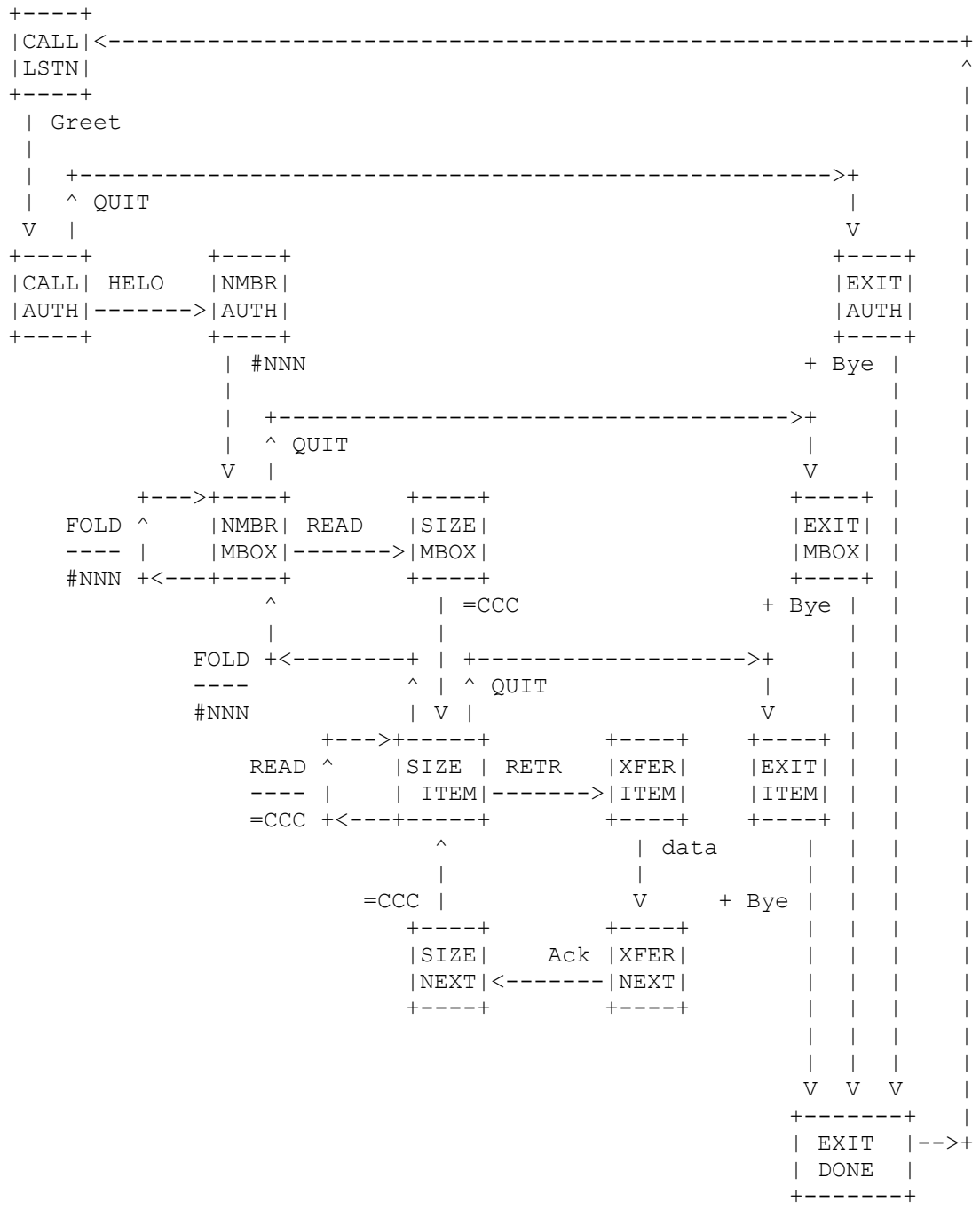
Client State Diagram



Server State Diagram



Combined Flow Diagram



RFC-937 Post Office Protocol - Version 2: Decision Tables

Client Decision Table

	STATE				
INPUT	CALL	NMBR	SIZE	XFER	EXIT
Greet	2	1	1	1	6
#NNN	1	3	1	1	6
=CCC	1	1	4	1	6
data	1	1	1	5	6
+ Bye	1	1	1	1	6
Close	1	1	1	1	6
other	1	1	1	1	6
Timeout	1	1	1	1	6

Actions:

1. This is garbage. Send "QUIT", and go to EXIT state.
2. (a) If the greeting is right then send "HELO" and go to NMBR state,

   (b) Else send "QUIT" and go to EXIT state.
3. (a) If user wants this folder and NNN > 0 then send "READ" and go to SIZE state,

   (b) If user wants a this folder and NNN = 0 then send "QUIT" and go to EXIT state,

   (c) If user wants a different folder then send "FOLD" and go to NMBR state.
4. (a) If user wants this message and CCC > 0 then send "RETR" and go to XFER state,

   (b) If user wants a this message and CCC = 0 then send "QUIT" and go to EXIT state,

   (c) If user wants a different message then send "READ" and go to SIZE state.
5. (a) If user wants this message kept then send "ACKS" and go to SIZE state,

   (b) If user wants a this message deleted then send "ACKD" and go to SIZE state,

   (c) If user wants a this message again then send "NACK" and go to SIZE state.
6. Close the connection.

RFC-937 Post Office Protocol - Version 2: Decision Tables

Server Decision Table

	STATE					
INPUT	LSTN	AUTH	MBOX	ITEM	NEXT	DONE
Open	2	1	1	1	1	1
HELO	1	3	1	1	1	1
FOLD	1	1	5	5	1	1
READ	1	1	6	6	1	1
RETR	1	1	1	7	1	1
ACKS	1	1	1	1	8	1
ACKD	1	1	1	1	8	1
NACK	1	1	1	1	8	1
QUIT	1	4	4	4	1	1
Close	1	1	1	1	1	9
other	1	1	1	1	1	1
Timeout		1	1	1	1	1

Actions:

1. This is garbage. Send "- error", and Close the connection.
2. Send the greeting. Go to AUTH state.
3. (a) If authorized user then send "#NNN" and go tp MBOX state,

   (b) Else send "- error" and Close the connection.
4. Send "+ Bye" and go to DONE state.
5. Send "+NNN" and go to MBOX state.
6. Send "=CCC" and go to ITEM state.
7. If message exists then send the data and got to NEXT state, Else Close the connection.
8. Do what ACKS/ACKD/NACK require and go to ITEM state.
9. Close the connection.

Acknowledgments

We would like to acknowledge the helpful comments that we received on the first version of POP described in RFC 918, and the draft of POP2 distributed to interested parties.

RFC-950 Internet Standard Subnetting Procedure

J. Mogul Stanford University
J. Postel USC/ISI
August 1985

Status Of This Memo

This RFC specifies a protocol for the ARPA-Internet community. As of May 1990 the IAB lists this protocol as a **required** standard. Distribution of this memo is unlimited.

RFC-950 Internet Standard Subnetting Procedure

Introduction

Overview

This memo discusses the utility of "subnets" of Internet networks, which are logically visible sub-sections of a single Internet network. For administrative or technical reasons, many organizations have chosen to divide one Internet network into several subnets, instead of acquiring a set of Internet network numbers. This memo specifies procedures for the use of subnets. These procedures are for hosts (e.g., workstations). The procedures used in and between subnet gateways are not fully described. Important motivation and background information for a subnetting standard is provided in "Toward an Internet Standard Scheme for Subnetting" [RFC-940].

Acknowledgment

This memo is based on "Internet Subnets" [RFC-917] and largely supercedes it. Many people contributed to the development of the concepts described here. J. Noel Chiappa, Chris Kent, and Tim Mann, in particular, provided important suggestions. Additional contributions in shaping this memo were made by Zaw-Sing Su, Mike Karels, and the Gateway Algorithms and Data Structures Task Force (GADS).

RFC-950 Internet Standard Subnetting Procedure

Motivation

The original view of the Internet universe was a two-level hierarchy: the top level the Internet as a whole, and the level below it individual networks, each with its own network number. The Internet does not have a hierarchical topology, rather the interpretation of addresses is hierarchical. In this two-level model, each host sees its network as a single entity; that is, the network may be treated as a "black box" to which a set of hosts is connected.

While this view has proved simple and powerful, a number of organizations have found it inadequate, and have added a third level to the interpretation of Internet addresses. In this view, a given Internet network is divided into a collection of subnets.

The three-level model is useful in networks belonging to moderately large organizations (e.g., Universities or companies with more than one building), where it is often necessary to use more than one LAN cable to cover a "local area". Each LAN may then be treated as a subnet.

There are several reasons why an organization might use more than one cable to cover a campus:

- Different technologies: Especially in a research environment, there may be more than one kind of LAN in use; e.g., an organization may have some equipment that supports Ethernet, and some that supports a ring network.
- Limits of technologies: Most LAN technologies impose limits, based on electrical parameters, on the number of hosts connected, and on the total length of the cable. It is easy to exceed these limits, especially those on cable length.
- Network congestion: It is possible for a small subset of the hosts on a LAN to monopolize most of the bandwidth. A common solution to this problem is to divide the hosts into cliques of high mutual communication, and put these cliques on separate cables.
- Point-to-Point links: Sometimes a "local area", such as a university campus, is split into two locations too far apart to connect using the preferred LAN technology. In this case, high-speed point-to-point links might connect several LANs.

An organization that has been forced to use more than one LAN has three choices for assigning Internet addresses:

1. Acquire a distinct Internet network number for each cable; subnets are not used at all.
2. Use a single network number for the entire organization, but assign host numbers without regard to which LAN a host is on ("transparent subnets").
3. Use a single network number, and partition the host address space by assigning subnet numbers to the LANs ("explicit subnets").

Each of these approaches has disadvantages. The first, although not requiring any new or modified protocols, results in an explosion in the size of Internet routing tables. Information about the internal details of local connectivity is propagated everywhere, although it is of

little or no use outside the local organization. Especially as some current gateway implementations do not have much space for routing tables, it would be good to avoid this problem.

The second approach requires some convention or protocol that makes the collection of LANs appear to be a single Internet network. For example, this can be done on LANs where each Internet address is translated to a hardware address using an Address Resolution Protocol (ARP), by having the bridges between the LANs intercept ARP requests for non-local targets, see "Multi-LAN Address Resolution" [RFC-925]. However, it is not possible to do this for all LAN technologies, especially those where ARP protocols are not currently used, or if the LAN does not support broadcasts. A more fundamental problem is that bridges must discover which LAN a host is on, perhaps by using a broadcast algorithm. As the number of LANs grows, the cost of broadcasting grows as well; also, the size of translation caches required in the bridges grows with the total number of hosts in the network.

The third approach is to explicitly support subnets. This does have a disadvantage, in that it is a modification of the Internet Protocol, and thus requires changes to IP implementations already in use (if these implementations are to be used on a subnetted network). However, these changes are relatively minor, and once made, yield a simple and efficient solution to the problem. Also, the approach avoids any changes that would be incompatible with existing hosts on non-subnetted networks.

Further, when appropriate design choices are made, it is possible for hosts which believe they are on a non-subnetted network to be used on a subnetted one, as explained in "Internet Subnets" [RFC-917]. This is useful when it is not possible to modify some of the hosts to support subnets explicitly, or when a gradual transition is preferred.

RFC-950 Internet Standard Subnetting Procedure

Standards for Subnet Addressing

This section first describes a proposal for interpretation of Internet addresses to support subnets. Next it discusses changes to host software to support subnets. Finally, it presents a procedures for discovering what address interpretation is in use on a given network (i.e., what address mask is in use).

RFC-950 Internet Standard Subnetting Procedure - Standards

Interpretation of Internet Addresses

Suppose that an organization has been assigned an Internet network number, has further divided that network into a set of subnets, and wants to assign host addresses: how should this be done? Since there are minimal restrictions on the assignment of the "local address" part of the Internet address, several approaches have been proposed for representing the subnet number:

1. Variable-width field: Any number of the bits of the local address part are used for the subnet number; the size of this field, although constant for a given network, varies from network to network. If the field width is zero, then subnets are not in use.
2. Fixed-width field: A specific number of bits (e.g., eight) is used for the subnet number, if subnets are in use.
3. Self-encoding variable-width field: Just as the width (i.e., class) of the network number field is encoded by its high-order bits, the width of the subnet field is similarly encoded.
4. Self-encoding fixed-width field: A specific number of bits is used for the subnet number.
5. Masked bits: Use a bit mask ("address mask") to identify which bits of the local address field indicate the subnet number.

What criteria can be used to choose one of these five schemes? First, should we use a self-encoding scheme? And, should it be possible to tell from examining an Internet address if it refers to a subnetted network, without reference to any other information?

An interesting feature of self-encoding is that it allows the address space of a network to be divided into subnets of different sizes, typically one subnet of half the address space and a set of small subnets.

For example, consider a class C network that uses a self-encoding scheme with one bit to indicate if it is the large subnet or not and an additional three bits to identify the small subnet. If the first bit is zero then this is the large subnet, if the first bit is one then the following bits (3 in this example) give the subnet number. There is one subnet with 128 host addresses, and eight subnets with 16 hosts each.

To establish a subnetting standard the parameters and interpretation of the self-encoding scheme must be fixed and consistent throughout the Internet.

It could be assumed that all networks are subnetted. This would allow addresses to be interpreted without reference to any other information.

This is a significant advantage, that given the Internet address no additional information is needed for an implementation to determine if two addresses are on the same subnet. However, this can also be viewed as a disadvantage: it may cause problems for networks which have existing host numbers that use arbitrary bits in the local address part. In other words, it is useful to be able to control whether a network is subnetted independently from the assignment of host addresses.

The alternative is to have the fact that a network is subnetted kept separate from the address. If one finds, somehow, that the network is subnetted then the standard

self-encoded subnetted network address rules are followed, otherwise the non-subnetted network addressing rules are followed.

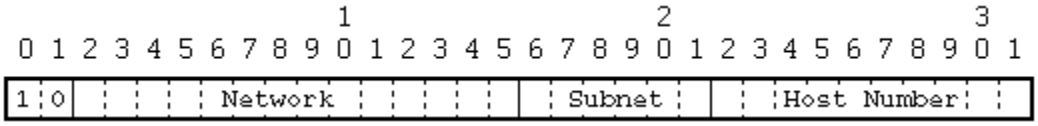
If a self-encoding scheme is not used, there is no reason to use a fixed-width field scheme: since there must in any case be some per-network "flag" to indicate if subnets are in use, the additional cost of using an integer (a subnet field width or address mask) instead of a boolean is negligible. The advantage of using the address mask scheme is that it allows each organization to choose the best way to allocate relatively scarce bits of local address to subnet and host numbers. Therefore, we choose the address-mask scheme: it is the most flexible scheme, yet costs no more to implement than any other.

For example, the Internet address might be interpreted as:

<network-number><subnet-number><host-number>

where the <network-number> field is as defined by the "Internet Protocol Specification" [RFC-791], the <host-number> field is at least 1-bit wide, and the width of the <subnet-number> field is constant for a given network. No further structure is required for the <subnet-number> or <host-number> fields. If the width of the <subnet-number> field is zero, then the network is not subnetted (i.e., the interpretation of the "Internet Protocol Specification" [RFC-791] is used).

For example, on a Class B network with a 6-bit wide subnet field, an address would be broken down like this:



Since the bits that identify the subnet are specified by a bitmask, they need not be adjacent in the address. However, we recommend that the subnet bits be contiguous and located as the most significant bits of the local address.

Special Addresses:

From the Assigned Numbers memo [RFC-1060]:

> "In certain contexts, it is useful to have fixed addresses with functional significance rather than as identifiers of specific hosts. When such usage is called for, the address zero is to be interpreted as meaning "this", as in "this network". The address of all ones are to be interpreted as meaning "all", as in "all hosts". For example, the address 128.9.255.255 could be interpreted as meaning all hosts on the network 128.9. Or, the address 0.0.0.37 could be interpreted as meaning host 37 on this network."

It is useful to preserve and extend the interpretation of these special addresses in subnetted networks. This means the values of all zeros and all ones in the subnet field should not be assigned to actual (physical) subnets.

> In the example above, the 6-bit wide subnet field may have any value except 0 and 63.

Please note that there is no effect or new restriction on the addresses of hosts on non-subnetted networks.

RFC-950 Internet Standard Subnetting Procedure - Standards

Changes to Host Software to Support Subnets

In most implementations of IP, there is code in the module that handles outgoing datagrams to decide if a datagram can be sent directly to the destination on the local network or if it must be sent to a gateway.

Generally the code is something like this:

```
IF ip_net_number(dg.ip_dest) = ip_net_number(my_ip_addr)
    THEN
        send_dg_locally(dg, dg.ip_dest)
    ELSE
        send_dg_locally(dg, gateway_to(ip_net_number(dg.ip_dest)))
```

(If the code supports multiply-connected networks, it will be more complicated, but this is irrelevant to the current discussion.)

To support subnets, it is necessary to store one more 32-bit quantity, called my_ip_mask. This is a bit-mask with bits set in the fields corresponding to the IP network number, and additional bits set corresponding to the subnet number field.

The code then becomes:

```
IF bitwise_and(dg.ip_dest, my_ip_mask)
                            = bitwise_and(my_ip_addr, my_ip_mask)
    THEN
        send_dg_locally(dg, dg.ip_dest)
    ELSE
        send_dg_locally(dg, gateway_to(bitwise_and(dg.ip_dest, my_ip_mask)))
```

Of course, part of the expression in the conditional can be pre-computed.

It may or may not be necessary to modify the "gateway_to" function, so that it too takes the subnet field bits into account when performing comparisons.

To support multiply-connected hosts, the code can be changed to keep the "my_ip_addr" and "my_ip_mask" quantities on a per-interface basis; the expression in the conditional must then be evaluated for each interface.

RFC-950 Internet Standard Subnetting Procedure - Standards

Finding the Address Mask

How can a host determine what address mask is in use on a subnet to which it is connected? The problem is analogous to several other "bootstrapping" problems for Internet hosts: how a host determines its own address, and how it locates a gateway on its local network. In all three cases, there are two basic solutions: "hardwired" information, and broadcast-based protocols.

Hardwired information is that available to a host in isolation from a network. It may be compiled-in, or (preferably) stored in a disk file. However, for the increasingly common case of a diskless workstation that is bootloaded over a LAN, neither hardwired solution is satisfactory.

Instead, since most LAN technology supports broadcasting, a better method is for the newly-booted host to broadcast a request for the necessary information. For example, for the purpose of determining its Internet address, a host may use the "Reverse Address Resolution Protocol (RARP)" [RFC-903].

However, since a newly-booted host usually needs to gather several facts (e.g., its IP address, the hardware address of a gateway, the IP address of a domain name server, the subnet address mask), it would be better to acquire all this information in one request if possible, rather than doing numerous broadcasts on the network. The mechanisms designed to boot diskless workstations can also load per-host specific configuration files that contain the required information (e.g., see "Bootstrap Protocol" [RFC-951]). It is possible, and desirable, to obtain all the facts necessary to operate a host from a boot server using only one broadcast message.

In the case where it is necessary for a host to find the address mask as a separate operation the following mechanism is provided:

> To provide the address mask information the ICMP protocol is extended by adding a new pair of ICMP message types, "Address Mask Request" and "Address Mask Reply", analogous to the "Information Request" and "Information Reply" ICMP messages.
>
> The intended use of these new ICMP messages is that a host, when booting, broadcast an "Address Mask Request" message. A gateway (or a host acting in lieu of a gateway) that receives this message responds with an "Address Mask Reply". If there is no indication in the request which host sent it (i.e., the IP Source Address is zero), the reply is broadcast as well. The requesting host will hear the response, and from it determine the address mask.
>
> Since there is only one possible value that can be sent in an "Address Mask Reply" on any given LAN, there is no need for the requesting host to match the responses it hears against the request it sent; similarly, there is no problem if more than one gateway responds. We assume that hosts reboot infrequently, so the broadcast load on a network from use of this protocol should be small.

If a host is connected to more than one LAN, it might have to find the address mask for each.

One potential problem is what a host should do if it can not find out the address mask, even after a reasonable number of tries. Three interpretations can be placed on the situation:

1. The local net exists in (permanent) isolation from all other nets.
2. Subnets are not in use, and no host can supply the address mask.
3. All gateways on the local net are (temporarily) down.

The first and second situations imply that the address mask is identical with the Internet network number mask. In the third situation, there is no way to determine what the proper value is; the safest choice is thus a mask identical with the Internet network number mask. Although this might later turn out to be wrong, it will not prevent transmissions that would otherwise succeed. It is possible for a host to recover from a wrong choice: when a gateway comes up, it should broadcast an "Address Mask Reply"; when a host receives such a message that disagrees with its guess, it should change its mask to conform to the received value. No host or gateway should send an "Address Mask Reply" based on a "guessed" value.

Finally, note that no host is required to use this ICMP protocol to discover the address mask; it is perfectly reasonable for a host with non-volatile storage to use stored information (including a configuration file from a boot server).

RFC-951 Bootstrap Protocol (BOOTP)

RFC-951 Bootstrap Protocol

Overview

BOOTP is an IP/UDP bootstrap protocol which allows a diskless client machine to discover its own IP address, the address of a server host, and the name of a file to be loaded into memory and executed. The bootstrap operation can be thought of as consisting of TWO PHASES. This describes the first phase, which could be labeled 'address determination and bootfile selection'. After this address and filename information is obtained, control passes to the second phase of the bootstrap where a file transfer occurs. The file transfer will typically use the TFTP protocol, since it is intended that both phases reside in PROM on the client. However BOOTP could also work with other protocols such as SFTP or FTP.

We suggest that the client's PROM software provide a way to do a complete bootstrap without 'user' interaction. This is the type of boot that would occur during an unattended power-up. A mechanism should be provided for the user to manually supply the necessary address and filename information to bypass the BOOTP protocol and enter the file transfer phase directly. If non-volatile storage is available, we suggest keeping default settings there and bypassing the BOOTP protocol unless these settings cause the file transfer phase to fail. If the cached information fails, the bootstrap should fall back to phase 1 and use BOOTP.

RFC-951 Bootstrap Protocol

Description

1. A single packet exchange is performed. Timeouts are used to retransmit until a reply is received. The same packet field layout is used in both directions. Fixed length fields of maximum reasonable length are used to simplify structure definition and parsing.
2. An 'opcode' field exists with two values. The client broadcasts a 'bootrequest' packet. The server then answers with a 'bootreply' packet. The bootrequest contains the client's hardware address and its IP address, if known.
3. The request can optionally contain the name of the server the client wishes to respond. This is so the client can force the boot to occur from a specific host (e.g. if multiple versions of the same bootfile exist or if the server is in a far distant net/domain). The client does not have to deal with name / domain services; instead this function is pushed off to the BOOTP server.
4. The request can optionally contain the 'generic' filename to be booted. For example 'unix' or 'ethertip'. When the server sends the bootreply, it replaces this field with the fully qualified path name of the appropriate boot file. In determining this name, the server may consult his own database correlating the client's address and filename request, with a particular boot file customized for that client. If the bootrequest filename is a null string, then the server returns a filename field indicating the 'default' file to be loaded for that client.
5. In the case of clients who do not know their IP addresses, the server must also have a database relating hardware address to IP address. This client IP address is then placed into a field in the bootreply.
6. Certain network topologies (such as Stanford's) may be such that a given physical cable does not have a TFTP server directly attached to it (e.g. all the gateways and hosts on a certain cable may be diskless). With the cooperation of neighboring gateways, BOOTP can allow clients to boot off of servers several hops away, through these gateways. See the section 'Booting Through Gateways' below. This part of the protocol requires no special action on the part of the client. Implementation is optional and requires a small amount of additional code in gateways and servers.

Packet Format

All numbers shown are decimal, unless indicated otherwise. The BOOTP packet is enclosed in a standard IP UDP datagram. For simplicity it is assumed that the BOOTP packet is never fragmented. Any numeric fields shown are packed in 'standard network byte order', i.e. high order bits are sent first.

In the IP header of a bootrequest, the client fills in its own IP source address if known, otherwise zero. When the server address is unknown, the IP destination address will be the 'broadcast address' 255.255.255.255. This address means 'broadcast on the local cable, (I don't know my net number)'.

The UDP header contains source and destination port numbers. The BOOTP protocol uses two reserved port numbers, 'BOOTP client' (68) and 'BOOTP server' (67). The client sends requests using 'BOOTP server' as the destination port; this is usually a broadcast. The server sends replies using 'BOOTP client' as the destination port; depending on the kernel or driver facilities in the server, this may or may not be a broadcast (this is explained further in 'Chicken/Egg issues'). The reason **two** reserved ports are used, is to avoid 'waking up' and scheduling the BOOTP server daemons, when a bootreply must be broadcast to a client. Since the server and other hosts won't be listening on the 'BOOTP client' port, any such incoming broadcasts will be filtered out at the kernel level. We could not simply allow the client to pick a 'random' port number for the UDP source port field; since the server reply may be broadcast, a randomly chosen port number could confuse other hosts that happened to be listening on that port.

The UDP length field is set to the length of the UDP plus BOOTP portions of the packet. The UDP checksum field can be set to zero by the client (or server) if desired, to avoid this extra overhead in a PROM implementation. See Field Bytes Description.

RFC-951 Bootstrap Protocol

Field Bytes Description

op 1	packet op code / message type. 1 = BOOTREQUEST, 2 = BOOTREPLY
htype 1	hardware address type, see ARP section in "Assigned Numbers" RFC. '1' = 10mb ethernet
hlen 1	hardware address length (eg '6' for 10mb ethernet).
hops 1	client sets to zero, optionally used by gateways in cross-gateway booting.
xid 4	transaction ID, a random number, used to match this boot request with the responses it generates.
secs 2	filled in by client, seconds elapsed since client started trying to boot.
ciaddr 4	client IP address; filled in by client in bootrequest if known.
yiaddr 4	'your' (client) IP address; filled by server if client doesn't know its own address (ciaddr was 0).
siaddr 4	server IP address; returned in bootreply by server.
giaddr 4	gateway IP address, used in optional cross-gateway booting.
chaddr 16	client hardware address, filled in by client.
sname 64	optional server host name, null terminated string.
file 128	boot file name, null terminated string; 'generic' name or null in bootrequest, fully qualified directory-path name in bootreply.
vend 64	optional vendor-specific area, e.g. could be hardware type/serial on request, or 'capability' / remote file system handle on reply. This info may be set aside for use by a third phase bootstrap or kernel.

Chicken / Egg Issues

How can the server send an IP datagram to the client, if the client doesnt know its own IP address (yet)? Whenever a bootreply is being sent, the transmitting machine performs the following operations:

1. If the client knows its own IP address ('ciaddr' field is nonzero), then the IP can be sent 'as normal', since the client will respond to ARPs [5].

2. If the client does not yet know its IP address (ciaddr zero), then the client cannot respond to ARPs sent by the transmitter of the bootreply. There are two options:

    a. If the transmitter has the necessary kernel or driver hooks to 'manually' construct an ARP address cache entry, then it can fill in an entry using the 'chaddr' and 'yiaddr' fields. Of course, this entry should have a timeout on it, just like any other entry made by the normal ARP code itself. The transmitter of the bootreply can then simply send the bootreply to the client's IP address. UNIX (4.2 BSD) has this capability.

    b. If the transmitter lacks these kernel hooks, it can simply send the bootreply to the IP broadcast address on the appropriate interface. This is only one additional broadcast over the previous case.

Client Use of ARP

The client PROM must contain a simple implementation of ARP, e.g. the address cache could be just one entry in size. This will allow a second-phase-only boot (TFTP) to be performed when the client knows the IP addresses and bootfile name.

Any time the client is expecting to receive a TFTP or BOOTP reply, it should be prepared to answer an ARP request for its own IP to hardware address mapping (if known).

Since the bootreply will contain (in the hardware encapsulation) the hardware source address of the server/gateway, the client MAY be able to avoid sending an ARP request for the server/gateway IP address to be used in the following TFTP phase. However this should be treated only as a special case, since it is desirable to still allow a second-phase-only boot as described above.

Comparison of BOOTP to RARP

An earlier protocol, Reverse Address Resolution Protocol (RARP) was established to allow a client to determine its IP address, given that it knew its hardware address. However RARP had the disadvantage that it was a hardware link level protocol (not IP/UDP based). This means that RARP could only be implemented on hosts containing special kernel or driver modifications to access these 'raw' packets. Since there are many network kernels existent now, with each source maintained by different organizations, a boot protocol that does not require kernel modifications is a decided advantage.

BOOTP provides this hardware to IP address lookup function, in addition to the other useful features described in the sections above.

Client Transmission

Before setting up the packet for the first time, it is a good idea to clear the entire packet buffer to all zeros; this will place all fields in their default state. The client then creates a packet with the following fields.

The IP destination address is set to 255.255.255.255. (the broadcast address) or to the server's IP address (if known). The IP source address and 'ciaddr' are set to the client's IP address if known, else 0. The UDP header is set with the proper length; source port = 'BOOTP client' port destination port = 'BOOTP server' port.

'op' is set to '1', BOOTREQUEST. 'htype' is set to the hardware address type as assigned in the ARP section of the "Assigned Numbers" RFC. 'hlen' is set to the length of the hardware address, e.g. '6' for 10mb ethernet.

'xid' is set to a 'random' transaction id. 'secs' is set to the number of seconds that have elapsed since the client has started booting. This will let the servers know how long a client has been trying. As the number gets larger, certain servers may feel more 'sympathetic' towards a client they don't normally service. If a client lacks a suitable clock, it could construct a rough estimate using a loop timer. Or it could choose to simply send this field as always a fixed value, say 100 seconds.

If the client knows its IP address, 'ciaddr' (and the IP source address) are set to this value. 'chaddr' is filled in with the client's hardware address.

If the client wishes to restrict booting to a particular server name, it may place a null-terminated string in 'sname'. The name used should be any of the allowable names or nicknames of the desired host.

The client has several options for filling the 'file' name field. If left null, the meaning is 'I want to boot the default file for my machine'. A null file name can also mean 'I am only interested in finding out client/server/gateway IP addresses, I dont care about file names'.

The field can also be a 'generic' name such as 'unix' or 'gateway'; this means 'boot the named program configured for my machine'. Finally the field can be a fully directory qualified path name.

The 'vend' field can be filled in by the client with vendor-specific strings or structures. For example the machine hardware type or serial number may be placed here. However the operation of the BOOTP server should not DEPEND on this information existing.

If the 'vend' field is used, it is recommended that a 4 byte 'magic number' be the first item within 'vend'. This lets a server determine what kind of information it is seeing in this field. Numbers can be assigned by the usual 'magic number' process --you pick one and it's magic. A different magic number could be used for bootreply's than bootrequest's to allow the client to take special action with the reply information.

A UDP checksum **should** be computed.

Client Retransmission Strategy

If no reply is received for a certain length of time, the client should retransmit the request. The time interval must be chosen carefully so as not to flood the network. Consider the case of a cable containing 100 machines that are just coming up after a power failure. Simply retransmitting the request every four seconds will inundate the net.

As a possible strategy, you might consider backing off exponentially, similar to the way ethernet backs off on a collision. So for example if the first packet is at time 0:00, the second would be at :04, then :08, then :16, then :32, then :64. You should also randomize each time; this would be done similar to the ethernet specification by starting with a mask and 'and'ing that with with a random number to get the first backoff. On each succeeding backoff, the mask is increased in length by one bit. This doubles the average delay on each backoff.

After the 'average' backoff reaches about 60 seconds, it should be increased no further, but still randomized.

Before each retransmission, the client should update the 'secs' field and recompute the UDP checksum.

Server Receives BOOTREQUEST

If the UDP destination port does not match the 'BOOTP server' port, discard the packet.

If the server name field (sname) is null (no particular server specified), or sname is specified and matches our name or nickname, then continue with packet processing.

If the sname field is specified, but does not match 'us', then there are several options:

1. You may choose to simply discard this packet.

2. If a name lookup on sname shows it to be on this same cable, discard the packet.

3. If sname is on a different net, you may choose to forward the packet to that address. If so, check the 'giaddr' (gateway address) field. If 'giaddr' is zero, fill it in with my address or the address of a gateway that can be used to get to that net. Then forward the packet.

If the client IP address (ciaddr) is zero, then the client does not know its own IP address. Attempt to lookup the client hardware address (chaddr, hlen, htype) in our database. If no match is found, discard the packet. Otherwise we now have an IP address for this client; fill it into the 'yiaddr' (your IP address) field.

We now check the boot file name field (file). The field will be null if the client is not interested in filenames, or wants the default bootfile. If the field is non-null, it is used as a lookup key in a database, along with the client's IP address. If there is a default file or generic file (possibly indexed by the client address) or a fully-specified path name that matches, then replace the 'file' field with the fully-specified path name of the selected boot file. If the field is non-null and no match was found, then the client is asking for a file we dont have; discard the packet, perhaps some other BOOTP server will have it.

The 'vend' vendor-specific data field should now be checked and if a recognized type of data is provided, client-specific actions should be taken, and a response placed in the 'vend' data field of the reply packet. For example, a workstation client could provide an authentication key and receive from the server a capability for remote file access, or a set of configuration options, which can be passed to the operating system that will shortly be booted in.

Place my (server) IP address in the 'siaddr' field. Set the 'op' field to BOOTREPLY. The UDP destination port is set to 'BOOTP client'. If the client address 'ciaddr' is nonzero, send the packet there; else if the gateway address 'giaddr' is nonzero, set the UDP destination port to 'BOOTP server' and send the packet to 'giaddr'; else the client is on one of our cables but it doesnt know its own IP address yet --use a method described in the 'Egg' section above to send it to the client. If 'Egg' is used and we have multiple interfaces on this host, use the 'yiaddr' (your IP address) field to figure out which net (cable/interface) to send the packet to. [UDP checksum.]

Server/Gateway Receives BOOTREPLY

If 'yiaddr' (your [the client's] IP address) refers to one of our cables, use one of the 'Egg' methods to forward it to the client. Be sure to send it to the 'BOOTP client' UDP destination port.

Client Reception

Don't forget to process ARP requests for my own IP address (if I know it). The client should discard incoming packets that: are not IP/UDPs addressed to the boot port; are not BOOTREPLYs; do not match my IP address (if I know it) or my hardware address; do not match my transaction id. Otherwise we have received a successful reply. 'yiaddr' will contain my IP address, if I didnt know it before. 'file' is the name of the file name to TFTP 'read request'. The server address is in 'siaddr'. If 'giaddr' (gateway address) is nonzero, then the packets should be forwarded there first, in order to get to the server.

Booting Through Gateways

This part of the protocol is optional and requires some additional code in cooperating gateways and servers, but it allows cross-gateway booting. This is mainly useful when gateways are diskless machines. Gateways containing disks (e.g. a UNIX machine acting as a gateway), might as well run their own BOOTP/TFTP servers.

Gateways listening to broadcast BOOTREQUESTs may decide to forward or rebroadcast these requests 'when appropriate'. For example, the gateway could have, as part of his configuration tables, a list of other networks or hosts to receive a copy of any broadcast BOOTREQUESTs. Even though a 'hops' field exists, it is a poor idea to simply globally rebroadcast the requests, since broadcast loops will almost certainly occur.

The forwarding could begin immediately, or wait until the 'secs' (seconds client has been trying) field passes a certain threshold.

If a gateway does decide to forward the request, it should look at the 'giaddr' (gateway IP address) field. If zero, it should plug its own IP address (on the receiving cable) into this field. It may also use the 'hops' field to optionally control how far the packet is reforwarded. Hops should be incremented on each forwarding. For example, if hops passes '3', the packet should probably be discarded.

Here we have recommended placing this special forwarding function in the gateways. But that does not have to be the case. As long as some 'BOOTP forwarding agent' exists on the net with the booting client, the agent can do the forwarding when appropriate. Thus this service may or may not be co-located with the gateway.

In the case of a forwarding agent not located in the gateway, the agent could save himself some work by plugging the broadcast address of the interface receiving the bootrequest into the 'giaddr' field. Thus the reply would get forwarded using normal gateways, not involving the forwarding agent. Of course the disadvantage here is that you lose the ability to use the 'Egg' non-broadcast method of sending the reply, causing extra overhead for every host on the client cable.

Sample Server Database

As a suggestion, we show a sample text file database that the BOOTP server program might use. The database has two sections, delimited by a line containing an percent in column 1. The first section contains a 'default directory' and mappings from generic names to directory/pathnames. The first generic name in this section is the 'default file' you get when the bootrequest contains a null 'file' string.

The second section maps hardware addresstype/address into an ipaddress. Optionally you can also overide the default generic name by supplying a ipaddress specific genericname. A 'suffix' item is also an option; if supplied, any generic names specified by the client will be accessed by first appending 'suffix' to the 'pathname' appropriate to that generic name. If that file is not found, then the plain 'pathname' will be tried. This 'suffix' option allows a whole set of custom generics to be setup without a lot of effort. Below is shown the general format; fields are delimited by one or more spaces or tabs; trailing empty fields may be omitted; blank lines and lines beginning with '#' are ignored.

```
# comment line
homedirectory
genericname1 pathname1
genericname2 pathname2
...
% end of generic names, start of address mappings
hostname1 hardwaretype hardwareaddr1 ipaddr1 genericname suffix
hostname2 hardwaretype hardwareaddr2 ipaddr2 genericname suffix
...
```

Here is a specific example. Note the 'hardwaretype' number is the same as that shown in the ARP section of the 'Assigned Numbers' RFC. The 'hardwaretype' and 'ipaddr' numbers are in decimal; 'hardwareaddr' is in hex.

```
# last updated by smith
/usr/boot
vmunix vmunix
tip ethertip
watch   /usr/diag/etherwatch
gategate.

% end of generic names, start of address mappings

hamilton   1 02.60.8c.06.34.98   36.19.0.5
```

```
burr1 02.60.8c.34.11.78   36.44.0.12
101-gateway   1 02.60.8c.23.ab.35   36.44.0.32gate 101
mjh-gateway   1 02.60.8c.12.32.bc   36.42.0.64gate mjh
welch-tipa1 02.60.8c.22.65.32   36.47.0.14tip
welch-tipb1 02.60.8c.12.15.c8   36.46.0.12tip
```

In the example above, if 'mjh-gateway' does a default boot, it will get the file '/usr/boot/gate.mjh'.

RFC-952 DoD Internet Host Table Specification

K. Harrenstien, M. Stahl, E. Feinler
SRI
October 1985

Status Of This Memo

This RFC is the official specification of the format of the Internet Host Table. Distribution of this memo is unlimited. Use of Host Tables in this format has been largely replaced by the Domain Name System as defined in RFC-1034 and RFC-1035.

RFC-952 DoD Internet Host Table Specification

Introduction

The DoD Host Table is utilized by the DoD Hostname Server maintained by the DDN Network Information Center (NIC) on behalf of the Defense Communications Agency (DCA) described in RFC-953.

Location Of The Standard Dod Online Host Table

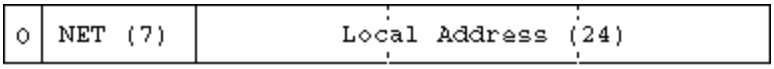
A machine-translatable ASCII text version of the DoD Host Table is online in the file NETINFO:HOSTS.TXT on the SRI-NIC host. It can be obtained via FTP from your local host by connecting to host SRI-NIC.ARPA (26.0.0.73 or 10.0.0.51), logging in as user = ANONYMOUS, password = GUEST, and retrieving the file "NETINFO:HOSTS.TXT". The same table may also be obtained via the NIC Hostname Server, as described in RFC-953. The latter method is faster and easier, but requires a user program to make the necessary connection to the Name Server.

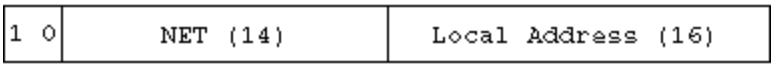
RFC-952 DoD Internet Host Table Specification

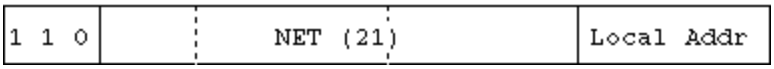
Assumptions

1. A "name" (Net, Host, Gateway, or Domain name) is a text string up to 24 characters drawn from the alphabet (A-Z), digits (0-9), minus sign (-), and period (.). Note that periods are only allowed when they serve to delimit components of "domain style names". ("Domain Name System", for background). No blank or space characters are permitted as part of a name. No distinction is made between upper and lower case. The first character must be an alpha character. The last character must not be a minus sign or period. A host which serves as a GATEWAY should have "-GATEWAY" or "-GW" as part of its name. Hosts which do not serve as Internet gateways should not use "-GATEWAY" and "-GW" as part of their names. A host which is a TAC should have "-TAC" as the last part of its host name, if it is a DoD host. Single character names or nicknames are not allowed.
2. Internet Addresses are 32-bit addresses as described in "Address Mappings" RFC-796. In the host table described herein each address is represented by four decimal numbers separated by a period. Each decimal number represents 1 octet.
3. If the first bit of the first octet of the address is 0 (zero), then the next 7 bits of the first octet indicate the network number (Class A Address). If the first two bits are 1,0 (one,zero), then the next 14 bits define the net number (Class B Address). If the first 3 bits are 1,1,0 (one,one,zero), then the next 21 bits define the net number (Class C Address) as described in the Internet Protocol Specification (RFC-791).

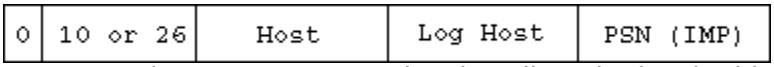
   This is depicted in the following diagram:







4. The LOCAL ADDRESS portion of the internet address identifies a host within the network specified by the NET portion of the address.
5. The ARPANET and MILNET are both Class A networks. The NET portion is 10 decimal for ARPANET, 26 decimal for MILNET, and the LOCAL ADDRESS maps as follows: the second octet identifies the physical host, the third octet identifies the logical host, and the fourth identifies the Packet Switching Node (PSN), formerly known as an Interface Message Processor (IMP).



   (NOTE: RFC-796 also describes the local address mappings for several other networks.)
6. It is the responsibility of the users of this host table to translate it into whatever format is needed for their purposes.

7. Names and addresses for DoD hosts and gateways will be negotiated and registered with the DDN PMO, and subsequently with the NIC, before being used and before traffic is passed by a DoD host. Names and addresses for domains and networks are to be registered with the DDN Network Information Center (HOSTMASTER@SRI-NIC.ARPA) or 800-235-3155.

The NIC will attempt to keep similar information for non-DoD networks and hosts, if this information is provided, and as long as it is needed, i.e., until intercommunicating network name servers are in place.

Example Of Host Table Format

```
NET : 10.0.0.0 : ARPANET :
NET : 128.10.0.0 : PURDUE-CS-NET :
GATEWAY : 10.0.0.77, 18.10.0.4 : MIT-GW.ARPA,MIT-GATEWAY : PDP-11 :
        MOS : IP/GW,EGP :
HOST : 26.0.0.73, 10.0.0.51 : SRI-NIC.ARPA,SRI-NIC,NIC : DEC-2060 :
        TOPS20 :TCP/TELNET,TCP/SMTP,TCP/TIME,TCP/FTP,TCP/ECHO,ICMP :
HOST : 10.2.0.11 : SU-TAC.ARPA,SU-TAC : C/30 : TAC : TCP :
```

RFC-952 DoD Internet Host Table Specification

Syntax And Conventions

; (semicolon) is used to denote the beginning of a comment. Any text on a given line following a ';' is a comment, and not part of the host table.

NET	keyword introducing a network entry
GATEWAY	keyword introducing a gateway entry
HOST	keyword introducing a host entry
DOMAIN	keyword introducing a domain entry
:(colon)	is used as a field delimiter
::(2 colons)	indicates a null field
,(comma)	is used as a data element delimiter
XXX/YYY	indicates protocol information of the type TRANSPORT/SERVICE. where TRANSPORT/SERVICE options are specified as: "FOO/BAR" both transport and service known; "FOO" transport known; services not known; "BAR" service is known, transport not known

NOTE: See "Assigned Numbers" for specific options and acronyms for machine types, operating systems, and protocol/services.

Each host table entry is an ASCII text string comprised of 6 fields, where

Field 1	KEYWORD indicating whether this entry pertains to a NET, GATEWAY, HOST, or DOMAIN. NET entries are assigned and cannot have alternate addresses or nicknames. DOMAIN entries do not use fields 4, 5, or 6.
Field 2	Internet Address of Network, Gateway, or Host followed by alternate addresses. Addresses for a Domain are those where a Domain Name Server exists for that domain.
Field 3	Official Name of Network, Gateway, Host, or Domain (with optional nicknames, where permitted).
Field 4	Machine Type
Field 5	Operating System
Field 6	Protocol List

Fields 4, 5 and 6 are optional. For a Domain they are not used.

Fields 3-6, if included, pertain to the first address in Field 2.

'Blanks' (spaces and tabs) are ignored between data elements or fields, but are disallowed within a data element.

Each entry ends with a colon.

The entries in the table are grouped by types in the order Domain, Net, Gateway, and Host. Within each type the ordering is unspecified.

Note that although optional nicknames are allowed for hosts, they are discouraged, except in the case where host names have been changed and both the new and the old names are maintained for a suitable period of time to effect a smooth transition. Nicknames are not permitted for NET names.

RFC-952 DoD Internet Host Table Specification

Grammatical Host Table Specification

A. Parsing grammar

<entry> ::= <keyword> ":" <addresses> ":" <names> [":" [<cputype>]
[":" [<opsys>] [":" [<protocol list>]]]] ":"
<addresses> ::= <address> *["," <address>]
<address> ::= <octet> "." <octet> "." <octet> "." <octet>
<octet> ::= <0 to 255 decimal>
<names> ::= <netname> | <gatename> | <domainname> *["," <nicknames>]
| <official hostname> *["," <nicknames>]
<netname> ::= <name>
<gatename> ::= <hname>
<domainname> ::= <hname>
<official hostname> ::= <hname>
<nickname> ::= <hname>
<protocol list> ::= <protocol spec> *["," <protocol spec>]
<protocol spec> ::= <transport name> "/" <service name>
| <raw protocol name>

B. Lexical grammar

<entry-field> ::= <entry-text> [<cr><lf> <blank> <entry-field>]
<entry-text> ::= <print-char> *<text>
<blank> ::= <space-or-tab> [<blank>]
<keyword> ::= NET | GATEWAY | HOST | DOMAIN
<hname> ::= <name>*["."<name>]
<name> ::= <let>[*[<let-or-digit-or-hyphen>]<let-or-digit>]
<cputype> ::= PDP-11/70 | DEC-1080 | C/30 | CDC-6400...etc.
<opsys> ::= ITS | MULTICS | TOPS20 | UNIX...etc.
<transport name> ::= TCP | NCP | UDP | IP...etc.
<service name> ::= TELNET | FTP | SMTP | MTP...etc.
<raw protocol name> ::= <name>
<comment> ::= ";" <text><cr><lf>
<text> ::= *[<print-char> | <blank>]
<print-char> ::= <any printing char (not space or tab)>

Notes:

1. Zero or more 'blanks' between separators " , : " are allowed. 'Blanks' are spaces and tabs.
2. Continuation lines are lines that begin with at least one blank. They may be used anywhere 'blanks' are legal to split an entry across lines.

RFC-953 Hostname Server

K Harrenstien, M. Stahl, E. Feinler
SRI
October 1985

Status Of This Memo

This RFC is the official specification of the Hostname Server Protocol. As of May 1990 the IAB lists this protocol as **elective**. Distribution of this memo is unlimited.

RFC-953 Hostname Server

Introduction

The NIC Internet Hostname Server is a TCP-based host information program and protocol running on the SRI-NIC machine. It is one of a series of internet name services maintained by the DDN Network Information Center (NIC) at SRI International on behalf of the Defense Communications Agency (DCA). The function of this particular server is to deliver machine-readable name/address information describing networks, gateways, hosts, and eventually domains, within the internet environment. As currently implemented, the server provides the information outlined in the DoD Internet Host Table Specification [RFC-952]. This protocol has been largely replaced by the Domain Name Service described in RFC-1034 and RFC-1035.

RFC-953 Hostname Server

Protocol

To access this server from a program, establish a TCP connection to port 101 (decimal) at the service host, SRI-NIC.ARPA (26.0.0.73 or 10.0.0.51). Send the information request (a single line), and read the resulting response. The connection is closed by the server upon completion of the response, so only one request can be made for each connection.

Query/Response Format

The name server accepts simple text query requests of the form:

<command key> <argument(s)> [<options>]

where square brackets ("[]") indicate an optional field. The command key is a keyword indicating the nature of the request. The defined keys are explained below.

The response, on the other hand, is of the form

<response key> : <rest of response>

where <response key> is a keyword indicating the nature of the response, and the rest of the response is interpreted in the context of the key.

NOTE: Care should be taken to interpret the nature of the reply (e.g, single record or multiple record), so that no confusion about the state of the reply results. An "ALL" request will likely return several hundred or more records of all types, whereas "HNAME" or "HADDR" will usually return one HOST record.

RFC-953 Hostname Server

Command/Response Keys

The currently defined command keywords are listed below. NOTE: Because the server and the features available will evolve with time, the HELP command should be used to obtain the most recent summary of implemented features, changes, or new commands.

Keyword	Response
HELP	This information.
VERSION	"VERSION: <string>" where <string> will be different for each version of the host table.
HNAME <hostname>	One or more matching host table entries.
HADDR <hostaddr>	One or more matching host table entries.
ALL	The entire host table.
ALL-OLD	The entire host table without domain style names.
DOMAINS	The entire top-level domain table (domains only).
ALL-DOM	Both the entire domain table and the host table.
ALL-INGWAY	All known gateways in TENEX/TOPS-20 INTERNET.GATEWAYS format.

Remember that the server accepts only a single command line and returns only a single response before closing the connection. HNAME and HADDR are useful for looking up a specific host by name or address; VERSION can be used by automated processes to see whether a "new" version of the host table exists without having to transfer the whole table. Note, however, that the returned version string is only guaranteed to be unique to each version, and nothing should currently be assumed about its format.

Response Keys:

ERR	entry not found, nature of error follows
NET	entry found, rest of entry follows
GATEWAY	entry found, rest of entry follows
HOST	entry found, rest of entry follows
DOMAIN	entry found, rest of entry follows
BEGIN	followed by multiple entries
END	done with BEGIN block of entries

More keywords will be added as new needs are recognized. A more detailed description of the allowed requests/responses follows.

Query/Response Examples

1. HNAME Query - Given a name, find the entry or entries that match the name. For example:

   HNAME SRI-NIC.ARPA <CRLF>

   where <CRLF> is a carriage return/ linefeed, and 'SRI-NIC.ARPA' is a host name. The likely response is:

   HOST : 26.0.0.73, 10.0.0.51 : SRI-NIC.ARPA,SRI-NIC,NIC :
   DEC-2060 : TOPS20 : TCP/TELNET,TCP/SMTP,TCP/TIME,TCP/FTP,
   TCP/ECHO,ICMP :

A response may stretch across more than one line. Continuation lines always begin with at least one space.

2. HADDR Query - Given an internet address (as specified in RFC 796) find the entry or entries that match that address. For example:

   HADDR 26.0.0.73 <CRLF>

   where <CRLF> is a carriage return/ linefeed, and '26.0.0.73' is a host address. The likely response is the same as for the previous HNAME request.

3. ALL Query - Deliver the entire internet host table in a machine-readable form. For example:

   ALL <CRLF>; where <CRLF> is a carriage return/linefeed

   The likely response is the keyword 'BEGIN' followed by a colon ':', followed by the entire internet host table in the format specified in RFC-952, followed by 'END:'.

Error Handling

ERR Reply - may occur on any query, and should be permitted in any access program using the name server. Errors are of the form

ERR : <code> : <string> :

as in

ERR : NAMNFD : Name not found :

The error code is a unique descriptor, limited to 8 characters in length for any given error. It may be used by the access program to identify the error and, in some cases, to handle it automatically. The string is an accompanying message for a given error for that case where the access program simply logs the error message. Current codes and their associated interpretations are:

NAMNFD	Name not found; name not in table
ADRNFD	Address not found; address not in table
ILLCOM	Illegal command; command key not recognized
TMPSYS	Temporary system failure, try again later

RFC-954 NICName/WhoIs

K. Harrenstien, M. Stahl, E. Feinler
SRI
October 1985

Status Of This Memo

This RFC is the official specification of the NICNAME/WHOIS protocol. This memo describes the protocol and the service. As of May 1990 the IAB lists this protocol as **elective**. Distribution of this memo is unlimited.

Introduction

The NICNAME/WHOIS Server is a TCP transaction based query/response server, running on the SRI-NIC machine (26.0.0.73 or 10.0.0.51), that provides netwide directory service to internet users. It is one of a series of internet name services maintained by the DDN Network Information Center (NIC) at SRI International on behalf of the Defense Communications Agency (DCA). The server is accessible across the Internet from user programs running on local hosts, and it delivers the full name, U.S. mailing address, telephone number, and network mailbox for DDN users who are registered in the NIC database.

This server, together with the corresponding WHOIS Database can also deliver online look-up of individuals or their online mailboxes, network organizations, DDN nodes and associated hosts, and TAC telephone numbers. The service is designed to be user-friendly and the information is delivered in human-readable format. DCA strongly encourages network hosts to provide their users with access to this network service.

Who Should Be In The Database

DCA requests that each individual with a directory on an ARPANET or MILNET host, who is capable of passing traffic across the DoD Internet, be registered in the NIC WHOIS Database. MILNET TAC users must be registered in the database. To register, send via electronic mail to REGISTRAR@SRI-NIC.ARPA your full name, middle initial, U.S. mailing address (including mail stop and full explanation of abbreviations and acronyms), ZIP code, telephone (including Autovon and FTS, if available), and one network mailbox. Contact the DDN Network Information Center, REGISTRAR@SRI-NIC.ARPA or (800) 235-3155, for assistance with registration.

Protocol

To access the NICNAME/WHOIS server:

Connect to the SRI-NIC service host at TCP service port 43 (decimal).

Send a single "command line", ending with <CRLF> (ASCII CR and LF).

Receive information in response to the command line. The server closes its connection as soon as the output is finished.

Existing User Programs

NICNAME is the global name for the user program, although many sites have chosen to use the more familiar name of "WHOIS". There are versions of the NICNAME user program for TENEX, TOPS-20, and UNIX. The TENEX and TOPS-20 programs are written in assembly language (FAIL/MACRO), and the UNIX version is written in C. They are easy to invoke, taking one argument which is passed directly to the NICNAME server at SRI-NIC. Contact NIC@SRI-NIC.ARPA for copies of the program.

Command Lines And Replies

A command line is normally a single name specification. Note that the specification formats will evolve with time; the best way to obtain the most recent documentation on name specifications is to give the server a command line consisting of "?<CRLF>" (that is, a question-mark alone as the name specification). The response from the NICNAME server will list all possible formats that can be used. The responses are not currently intended to be machine-readable; the information is meant to be passed back directly to a human user. The following three examples illustrate the use of NICNAME as of October 1985.

Command line: ?

Response:

Please enter a name or a NIC handle, such as "Smith" or "SRI-NIC". Starting with a period forces a name-only search; starting with exclamation point forces handle-only. Examples:

```
Smith          [looks for name or handle SMITH]
!SRI-NIC       [looks for handle SRI-NIC only]
.Smith, John [looks for name JOHN SMITH only]
```

Adding "..." to the argument will match anything from that point, e.g. "ZU..." will match ZUL, ZUM, etc.

To search for mailboxes, use one of these forms:

```
Smith@         [looks for mailboxes with username SMITH]
@Host          [looks for mailboxes on HOST]
Smith@Host [Looks for mailboxes with username SMITH on HOST]
```

To obtain the entire membership list of a group or organization, or a list of all authorized users of a host, precede the name of the host or organization by an asterisk, i.e. *SRI-NIC. [CAUTION: If there are a lot of members, this will take a long time!] You may use exclamation point and asterisk, or a period and asterisk together.

Command line: fischer

Response:

Fischer, Charles (CF17)	fischer@UWISC	(608) 262-1204
Fischer, Herman (HF)	HFischer@USC-ECLB	(818) 902-5139
Fischer, Jeffery H. (JHF1)	FISCHER@LL-XN	(617) 863-5500 ext 4403 or 4689
Fischer, Kenneth (KF8)	SAC.SIUBO@USC-ISIE	(402) 294-5161 (AV) 271-5161
Fischer, Marty (MF28)	MFISCHER@DCA-EMS	(703) 437-2344
Fischer, Michael J. (MJF)	FISCHER@YALE	(203) 436-0744
Fischer, Nancy C. (NANCY)	FISCHER@SRI-NIC	(415) 859-2539
Fischer, Richard A. (RAF4)	Fisher Richa@LLL-MFE	(415) 422-5032

To single out any individual entry, repeat the command using the

argument "!HANDLE" instead of "NAME", where the handle is in parentheses following the name.

Command line: !nancy

Response:

Fischer, Nancy C. (NANCY) FISCHER@SRI-NIC SRI International
Telecommunication Sciences Center
333 Ravenswood Avenue, EJ289
Menlo Park, California 94025
Phone: (415) 859-2539
MILNET TAC user

RFC-959 File Transfer Protocol (FTP)

J. Postel & J. Reynolds
USC/Information Sciences Instiitute
October 1985

Status of this Memo

This memo is the official specification of the File Transfer Protocol (FTP). Distribution of this memo is unlimited.

This version of the specification includes the original text plus additions and corrections that have been made since its publication.

RFC-959 File Transfer Protocol (FTP)

Introduction

The objectives of FTP are 1) to promote sharing of files (computer programs and/or data), 2) to encourage indirect or implicit (via programs) use of remote computers, 3) to shield a user from variations in file storage systems among hosts, and 4) to transfer data reliably and efficiently. FTP, though usable directly by a user at a terminal, is designed mainly for use by programs.

The attempt in this specification is to satisfy the diverse needs of users of maxi-hosts, mini-hosts, personal workstations, and TACs, with a simple, and easily implemented protocol design.

This paper assumes knowledge of the Transmission Control Protocol (TCP) as described in RFC-793 and the Telnet Protocol as described in RFC-854.

RFC-959 File Transfer Protocol (FTP)

Overview

In this section, the history, the terminology, and the FTP model are discussed. The terms defined in this section are only those that have special significance in FTP. Some of the terminology is very specific to the FTP model; some readers may wish to turn to the section on the FTP model while reviewing the terminology.

RFC-959 File Transfer Protocol (FTP) -- Overview

History

FTP has had a long evolution over the years and has evolved through a long series of RFCS. These original specifications are not included in this document, nor are the complete bibliographic references, since the current specification supercedes all of them.

These early specifications include the first proposed file transfer mechanisms in 1971 that were developed for implementation on hosts at M.I.T. (RFC 114), plus comments and discussion in RFC 141.

RFC 172 provided a user-level oriented protocol for file transfer between host computers (including terminal IMPs). A revision of this as RFC 265, restated FTP for additional review, while RFC 281 suggested further changes. The use of a "Set Data Type" transaction was proposed in RFC 294 in January 1982.

RFC 354 obsoleted RFCs 264 and 265. The File Transfer Protocol was now defined as a protocol for file transfer between HOSTs on the ARPANET, with the primary function of FTP defined as transfering files efficiently and reliably among hosts and allowing the convenient use of remote file storage capabilities. RFC 385 further commented on errors, emphasis points, and additions to the protocol, while RFC 414 provided a status report on the working server and user FTPs. RFC 430, issued in 1973, (among other RFCs too numerous to mention) presented further comments on FTP. Finally, an "official" FTP document was published as RFC 454.

By July 1973, considerable changes from the last versions of FTP were made, but the general structure remained the same. RFC 542 was published as a new "official" specification to reflect these changes. However, many implementations based on the older specification were not updated.

In 1974, RFCs 607 and 614 continued comments on FTP. RFC 624 proposed further design changes and minor modifications. In 1975, RFC 686 entitled, "Leaving Well Enough Alone", discussed the differences between all of the early and later versions of FTP. RFC 691 presented a minor revision of RFC 686, regarding the subject of print files.

Motivated by the transition from the NCP to the TCP as the underlying protocol, a phoenix was born out of all of the above efforts in RFC 765 as the specification of FTP for use on TCP.

This current edition of the FTP specification is intended to correct some minor documentation errors, to improve the explanation of some protocol features, and to add some new optional commands.

In particular, the following new commands are included in this edition of the specification:

CDUP - Change to Parent Directory
SMNT - Structure Mount
STOU - Store Unique
RMD - Remove Directory
MKD - Make Directory
PWD - Print Directory
SYST - System

This specification is compatible with the previous edition. A program implemented in conformance to the previous specification should automatically be in conformance to this specification.

Terminology

ASCII

The ASCII character set is as defined in the ARPA-Internet Protocol Handbook. In FTP, ASCII characters are defined to be the lower half of an eight-bit code set (i.e., the most significant bit is zero).

access controls

Access controls define users' access privileges to the use of a system, and to the files in that system. Access controls are necessary to prevent unauthorized or accidental use of files. It is the prerogative of a server-FTP process to invoke access controls.

byte size

There are two byte sizes of interest in FTP: the logical byte size of the file, and the transfer byte size used for the transmission of the data. The transfer byte size is always 8 bits. The transfer byte size is not necessarily the byte size in which data is to be stored in a system, nor the logical byte size for interpretation of the structure of the data.

control connection

The communication path between the USER-PI and SERVER-PI for the exchange of commands and replies. This connection follows the Telnet Protocol.

data connection

A full duplex connection over which data is transferred, in a specified mode and type. The data transferred may be a part of a file, an entire file or a number of files. The path may be between a server-DTP and a user-DTP, or between two server-DTPs.

data port

The passive data transfer process "listens" on the data port for a connection from the active transfer process in order to open the data connection.

DTP

The data transfer process establishes and manages the data connection. The DTP can be passive or active.

End-of-Line

The end-of-line sequence defines the separation of printing lines. The sequence is Carriage Return, followed by Line Feed.

EOF

The end-of-file condition that defines the end of a file being transferred.

EOR

The end-of-record condition that defines the end of a record being transferred.

error recovery

A procedure that allows a user to recover from certain errors such as failure of

either host system or transfer process. In FTP, error recovery may involve restarting a file transfer at a given checkpoint.

FTP commands

A set of commands that comprise the control information flowing from the user-FTP to the server-FTP process.

file

An ordered set of computer data (including programs), of arbitrary length, uniquely identified by a pathname.

mode

The mode in which data is to be transferred via the data connection. The mode defines the data format during transfer including EOR and EOF. The transfer modes defined in FTP are described in the Section on Transmission Modes.

NVT

The Network Virtual Terminal as defined in the Telnet Protocol.

NVFS

The Network Virtual File System. A concept which defines a standard network file system with standard commands and pathname conventions.

page

A file may be structured as a set of independent parts called pages. FTP supports the transmission of discontinuous files as independent indexed pages.

pathname

Pathname is defined to be the character string which must be input to a file system by a user in order to identify a file. Pathname normally contains device and/or directory names, and file name specification. FTP does not yet specify a standard pathname convention. Each user must follow the file naming conventions of the file systems involved in the transfer.

PI

The protocol interpreter. The user and server sides of the protocol have distinct roles implemented in a user-PI and a server-PI.

record

A sequential file may be structured as a number of contiguous parts called records. Record structures are supported by FTP but a file need not have record structure.

reply

A reply is an acknowledgment (positive or negative) sent from server to user via the control connection in response to FTP commands. The general form of a reply is a completion code (including error codes) followed by a text string. The codes are for use by programs and the text is usually intended for human users.

server-DTP

The data transfer process, in its normal "active" state, establishes the data connection with the "listening" data port. It sets up parameters for transfer

and storage, and transfers data on command from its PI. The DTP can be placed in a "passive" state to listen for, rather than initiate a connection on the data port.

server-FTP process

A process or set of processes which perform the function of file transfer in cooperation with a user-FTP process and, possibly, another server. The functions consist of a protocol interpreter (PI) and a data transfer process (DTP).

server-PI

The server protocol interpreter "listens" on Port L for a connection from a user-PI and establishes a control communication connection. It receives standard FTP commands from the user-PI, sends replies, and governs the server-DTP.

type

The data representation type used for data transfer and storage. Type implies certain transformations between the time of data storage and data transfer. The representation types defined in FTP are described in the Section on Establishing Data Connections.

user

A person or a process on behalf of a person wishing to obtain file transfer service. The human user may interact directly with a server-FTP process, but use of a user-FTP process is preferred since the protocol design is weighted towards automata.

user-DTP

The data transfer process "listens" on the data port for a connection from a server-FTP process. If two servers are transferring data between them, the user-DTP is inactive.

user-FTP process

A set of functions including a protocol interpreter, a data transfer process and a user interface which together perform the function of file transfer in cooperation with one or more server-FTP processes. The user interface allows a local language to be used in the command-reply dialogue with the user.

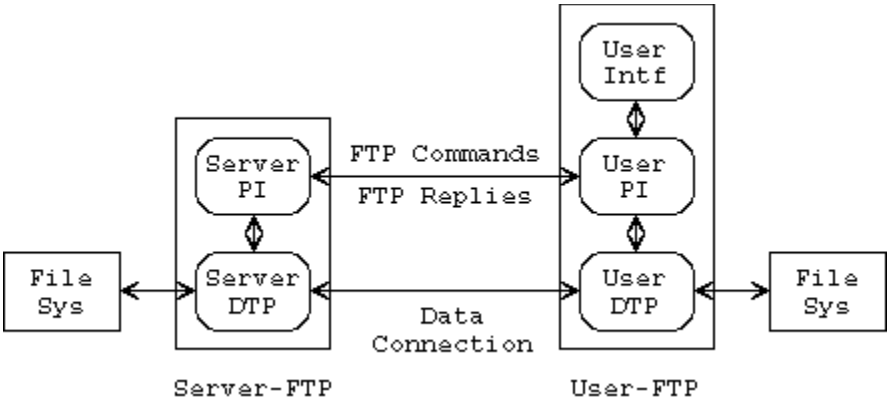
user-PI

The user protocol interpreter initiates the control connection from its port U to the server-FTP process, initiates FTP commands, and governs the user-DTP if that process is part of the file transfer.

RFC-959 File Transfer Protocol (FTP) -- Overview

The FTP Model

With the above definitions in mind, the following model (shown in Figure 1) may be diagrammed for an FTP service.



NOTES: 1. The data connection may be used in either direction.
2. The data connection need not exist all of the time.

Figure 1 Model for FTP Use

In the model described in Figure 1, the user-protocol interpreter initiates the control connection. The control connection follows the Telnet protocol. At the initiation of the user, standard FTP commands are generated by the user-PI and transmitted to the server process via the control connection. (The user may establish a direct control connection to the server-FTP, from a TAC terminal for example, and generate standard FTP commands independently, bypassing the user-FTP process.) Standard replies are sent from the server-PI to the user-PI over the control connection in response to the commands.

The FTP commands specify the parameters for the data connection (data port, transfer mode, representation type, and structure) and the nature of file system operation (store, retrieve, append, delete, etc.). The user-DTP or its designate should "listen" on the specified data port, and the server initiate the data connection and data transfer in accordance with the specified parameters. It should be noted that the data port need not be in the same host that initiates the FTP commands via the control connection, but the user or the user-FTP process must ensure a "listen" on the specified data port. It ought to also be noted that the data connection may be used for simultaneous sending and receiving.

In another situation a user might wish to transfer files between two hosts, neither of which is a local host. The user sets up control connections to the two servers and then arranges for a data connection between them. In this manner, control information is passed to the user-PI but data is transferred between the server data transfer processes. Following is a model of this server-server interaction.

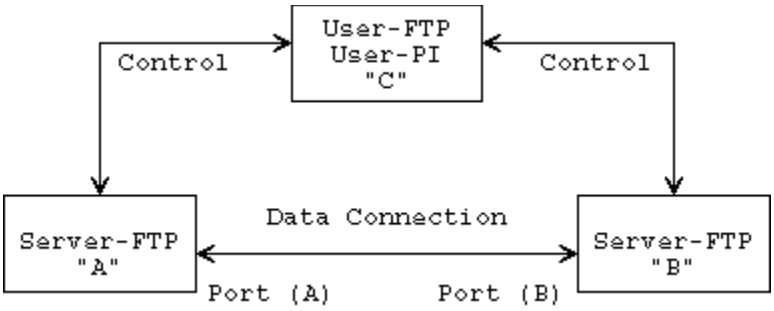


Figure 2

The protocol requires that the control connections be open while data transfer is in progress. It is the responsibility of the user to request the closing of the control connections when finished using the FTP service, while it is the server who takes the action. The server may abort data transfer if the control connections are closed without command.

The Relationship between FTP and Telnet:

The FTP uses the Telnet protocol on the control connection. This can be achieved in two ways: first, the user-PI or the server-PI may implement the rules of the Telnet Protocol directly in their own procedures; or, second, the user-PI or the server-PI may make use of the existing Telnet module in the system.

Ease of implementaion, sharing code, and modular programming argue for the second approach. Efficiency and independence argue for the first approach. In practice, FTP relies on very little of the Telnet Protocol, so the first approach does not necessarily involve a large amount of code.

Data Transfer Functions

Files are transferred only via the data connection. The control connection is used for the transfer of commands, which describe the functions to be performed, and the replies to these commands. Several commands are concerned with the transfer of data between hosts. These data transfer commands include the MODE command which specify how the bits of the data are to be transmitted, and the STRUcture and TYPE commands, which are used to define the way in which the data are to be represented. The transmission and representation are basically independent but the "Stream" transmission mode is dependent on the file structure attribute and if "Compressed" transmission mode is used, the nature of the filler byte depends on the representation type.

Data Representation And Storage

Data is transferred from a storage device in the sending host to a storage device in the receiving host. Often it is necessary to perform certain transformations on the data because data storage representations in the two systems are different. For example, NVT-ASCII has different data storage representations in different systems. DEC TOPS-20s's generally store NVT-ASCII as five 7-bit ASCII characters, left-justified in a 36-bit word. IBM Mainframe's store NVT-ASCII as 8-bit EBCDIC codes. Multics stores NVT-ASCII as four 9-bit characters in a 36-bit word. It is desirable to convert characters into the standard NVT-ASCII representation when transmitting text between dissimilar systems. The sending and receiving sites would have to perform the necessary transformations between the standard representation and their internal representations.

A different problem in representation arises when transmitting binary data (not character codes) between host systems with different word lengths. It is not always clear how the sender should send data, and the receiver store it. For example, when transmitting 32-bit bytes from a 32-bit word-length system to a 36-bit word-length system, it may be desirable (for reasons of efficiency and usefulness) to store the 32-bit bytes right-justified in a 36-bit word in the latter system. In any case, the user should have the option of specifying data representation and transformation functions. It should be noted that FTP provides for very limited data type representations. Transformations desired beyond this limited capability should be performed by the user directly.

Data Types
Data Structures

RFC-959 File Transfer Protocol (FTP) -- Data Representation and Storage

Data Types

Data representations are handled in FTP by a user specifying a representation type. This type may implicitly (as in ASCII or EBCDIC) or explicitly (as in Local byte) define a byte size for interpretation which is referred to as the "logical byte size." Note that this has nothing to do with the byte size used for transmission over the data connection, called the "transfer byte size", and the two should not be confused. For example, NVT-ASCII has a logical byte size of 8 bits. If the type is Local byte, then the TYPE command has an obligatory second parameter specifying the logical byte size. The transfer byte size is always 8 bits.

ASCII Type (A)

This is the default type and **must** be accepted by all FTP implementations. It is intended primarily for the transfer of text files, except when both hosts would find the EBCDIC type more convenient.

The sender converts the data from an internal character representation to the standard 8-bit NVT-ASCII representation (see the Telnet specification). The receiver will convert the data from the standard form to his own internal form.

In accordance with the NVT standard, the <CRLF> sequence should be used where necessary to denote the end of a line of text. (See the discussion of file structure at the end of the Section on Data Representation and Storage.)

Using the standard NVT-ASCII representation means that data must be interpreted as 8-bit bytes.

EBCDIC Type (E)

This type is intended for efficient transfer between hosts which use EBCDIC for their internal character representation.

For transmission, the data are represented as 8-bit EBCDIC characters. The character code is the only difference between the functional specifications of EBCDIC and ASCII types.

End-of-line (as opposed to end-of-record--see the discussion of structure) will probably be rarely used with EBCDIC type for purposes of denoting structure, but where it is necessary the <NL> character should be used.

IMAGE Type (I)

The data are sent as contiguous bits which, for transfer, are packed into the 8-bit transfer bytes. The receiving site must store the data as contiguous bits. The structure of the storage system might necessitate the padding of the file (or of each record, for a record-structured file) to some convenient boundary (byte, word or block). This padding, which must be all zeros, may occur only at the end of the file (or at the end of each record) and there must be a way of identifying the padding bits so that they may be stripped off if the file is retrieved. The padding transformation should be well publicized to enable a user to process a file at the storage site.

Image type is intended for the efficient storage and retrieval of files and for the transfer of binary data. An FTP program **must** support this type.

LOCAL Type (L)

The data is transferred in logical bytes of the size specified by the obligatory second parameter, Byte size. The value of Byte size must be a decimal integer; there is no default value. The logical byte size is not necessarily the same as the transfer byte size. If there is a difference in byte sizes, then the logical bytes should be packed contiguously, disregarding transfer byte boundaries and with any necessary padding at the end.

When the data reaches the receiving host, it will be transformed in a manner dependent on the logical byte size and the particular host. This transformation must be invertible (i.e., an identical file can be retrieved if the same parameters are used) and should be well publicized by the FTP implementors.

For example, a user sending 36-bit floating-point numbers to a host with a 32-bit word could send that data as Local byte with a logical byte size of 36. The receiving host would then be expected to store the logical bytes so that they could be easily manipulated; in this example putting the 36-bit logical bytes into 64-bit double words should suffice.

In another example, a pair of hosts with a 36-bit word size may send data to one another in words by using TYPE L 36. The data would be sent in the 8-bit transmission bytes packed so that 9 transmission bytes carried two host words.

An FTP program **must** support TYPE L 8. A machine whose memory is organized into m-bit words, where m is not a multiple of 8, **may** also support TYPE L m.

> **Discussion**
> The command "TYPE L 8" is often required to transfer binary data between a machine whose memory is organized into (e.g.) 36-bit words and a machinewith an 8-bit byte organization. For an 8-bit byte machine, TYPE L 8 is equivalent to IMAGE.
>
> "TYPE L m" is sometimes specified to the FTP programs on two m-bit word machines to ensure the correct transfer of a native-mode binary file from one machine to the other. However, this command should have the same effect on these machines as "TYPE I".

Format Control

The types ASCII and EBCDIC also take a second (optional) parameter; this is to indicate what kind of vertical format control, if any, is associated with a file. The following data representation types are defined in FTP:

A character file may be transferred to a host for one of three purposes: for printing, for storage and later retrieval, or for processing. If a file is sent for printing, the receiving host must know how the vertical format control is represented. In the second case, it must be possible to store a file at a host and then retrieve it later in exactly the same form. Finally, it should be possible to move a file from one host to another and process the file at the second host without undue trouble. A single ASCII or EBCDIC format does not satisfy all these conditions. Therefore, these types have a second parameter specifying one of the following three formats:

Non Print (N)

This is the default format to be used if the second (format) parameter is omitted. Non-print format must be accepted by all FTP implementations. The file need contain no vertical format information. If it is passed to a printer process, this process may assume standard values for spacing and margins.

Normally, this format will be used with files destined for processing or just storage.

Telnet Format Controls (T)

The file contains ASCII/EBCDIC vertical format controls (i.e., <CR>, <LF>, <NL>, <VT>, <FF>) which the printer process will interpret appropriately. <CRLF>, in exactly this sequence, also denotes end-of-line.

A host that makes no distinction between TYPE N and TYPE T **should** implement TYPE T to be identical to TYPE N.

> **Discussion**
> This provision should ease interoperation with hosts that do make this distinction.
>
> Many hosts represent text files internally as strings of ASCII characters, using the embedded ASCII format effector characters (LF, BS, FF, ...) to control theformat when a file is printed. For such hosts, there is no distinction between "print" files and other files. However, systems that use record structured files typically need a special format for printable files (e.g., ASA carriage control). For the latter hosts, FTP allows a choice of TYPE N or TYPE T.

Carriage Control (C)

The file contains ASA (FORTRAN) vertical format control characters. (See RFC 740 Appendix C; and Communications of the ACM, Vol. 7, No. 10, p. 606, October 1964.) In a line or a record formatted according to the ASA Standard, the first character is not to be printed. Instead, it should be used to determine the vertical movement of the paper which should take place before the rest of the record is printed.

The ASA Standard specifies the following control characters:

Character	Vertical Spacing
blank	Move paper up one line
0	Move paper up two lines
1	Move paper to top of next page
+	No movement, i.e., overprint

Clearly there must be some way for a printer process to distinguish the end of the structural entity. If a file has record structure (see below) this is no problem; records will be explicitly marked during transfer and storage. If the file has no record structure, the <CRLF> end-of-line sequence is used to separate printing lines, but these format effectors are overridden by the ASA controls.

Data Structures

In addition to different representation types, FTP allows the structure of a file to be specified. Three file structures are defined in FTP:

file-structure where there is no internal structure and the file is considered to be a continuous sequence of data bytes,

record-structure where the file is made up of sequential records,

page-structure where the file is made up of independent indexed pages.

File-structure is the default to be assumed if the STRUcture command has not been used but both file and record structures must be accepted for "text" files (i.e., files with TYPE ASCII or EBCDIC) by all FTP implementations. The structure of a file will affect both the transfer mode of a file (see the Section on Transmission Modes) and the interpretation and storage of the file.

The "natural" structure of a file will depend on which host stores the file. A source-code file will usually be stored on an IBM Mainframe in fixed length records but on a DEC TOPS-20 as a stream of characters partitioned into lines, for example by <CRLF>. If the transfer of files between such disparate sites is to be useful, there must be some way for one site to recognize the other's assumptions about the file.

With some sites being naturally file-oriented and others naturally record-oriented there may be problems if a file with one structure is sent to a host oriented to the other. If a text file is sent with record-structure to a host which is file oriented, then that host should apply an internal transformation to the file based on the record structure.

An FTP transformation between record-structure and file- structure **should** be invertible, to the extent possible while making the result useful on the target host.

> **Discussion**
> There are two different objectives for transferring a file: processing it on the target host, or just storage. For storage, strict invertibility is important. For processing, the file created on the target host needs to be in the format expected by application programs on that host.
>
> As an example of the conflict, imagine a record- oriented operating system that requires some data files to have exactly 80 bytes in each record. While STORing a file on such a host, an FTP Server must be able to pad each line or record to 80 bytes; a later retrieval of such a file cannot be strictly invertible.

In the case of a file being sent with file-structure to a record-oriented host, there exists the question of what criteria the host should use to divide the file into records which can be processed locally. If this division is necessary, the FTP implementation should use the end-of-line sequence, <CRLF> for ASCII, or <NL> for EBCDIC text files, as the delimiter. If an FTP implementation adopts this technique, it must be prepared to reverse the transformation if the file is retrieved with file-structure.

File Structure

File structure is the default to be assumed if the STRUcture command has not been used.

In file-structure there is no internal structure and the file is considered to be a continuous sequence of data bytes.

Record Structure

Record structures must be accepted for "text" files (i.e., files with TYPE ASCII or EBCDIC) by all FTP implementations.

In record-structure the file is made up of sequential records.

Page Structure

Implementation of page structure is **not recommended** in general. However, if a host system does need to implement FTP for "random access" or "holey" files, it **must** use the defined page structure format rather than define a new private FTP format.

To transmit files that are discontinuous, FTP defines a page structure. Files of this type are sometimes known as "random access files" or even as "holey files". In these files there is sometimes other information associated with the file as a whole (e.g., a file descriptor), or with a section of the file (e.g., page access controls), or both. In FTP, the sections of the file are called pages.

To provide for various page sizes and associated information, each page is sent with a page header. The page header has the following defined fields:

Header Length

The number of logical bytes in the page header including this byte. The minimum header length is 4.

Page Index

The logical page number of this section of the file. This is not the transmission sequence number of this page, but the index used to identify this page of the file.

Data Length

The number of logical bytes in the page data. The minimum data length is 0.

Page Type

The type of page this is. The following page types are defined:

0 = Last Page

This is used to indicate the end of a paged structured transmission. The header length must be 4, and the data length must be 0.

1 = Simple Page

This is the normal type for simple paged files with no page level associated control information. The header length must be 4.

2 = Descriptor Page

This type is used to transmit the descriptive information for the file as a whole.

3 = Access Controlled Page

This type includes an additional header field for paged files with page level access control information. The header length must be 5.

Optional Fields

Further header fields may be used to supply per page control information, for example, per page access control.

All fields are one logical byte in length. The logical byte size is specified by the TYPE command. See Appendix I for further details and a specific case at the page structure.

A note of caution about parameters: a file must be stored and retrieved with the same parameters if the retrieved version is to be identical to the version originally transmitted. Conversely, FTP implementations must return a file identical to the original if the parameters used to store and retrieve a file are the same.

RFC-959 File Transfer Protocol (FTP) -- Data Transfer Functions

Establishing Data Connections

The mechanics of transferring data consists of setting up the data connection to the appropriate ports and choosing the parameters for transfer. Both the user and the server-DTPs have a default data port. The user-process default data port is the same as the control connection port (i.e., U). The server-process default data port is the port adjacent to the control connection port (i.e., L-1).

The transfer byte size is 8-bit bytes. This byte size is relevant only for the actual transfer of the data; it has no bearing on representation of the data within a host's file system.

The passive data transfer process (this may be a user-DTP or a second server-DTP) shall "listen" on the data port prior to sending a transfer request command. The FTP request command determines the direction of the data transfer. The server, upon receiving the transfer request, will initiate the data connection to the port. When the connection is established, the data transfer begins between DTP's, and the server-PI sends a confirming reply to the user-PI.

Every FTP implementation must support the use of the default data ports, and only the USER-PI can initiate a change to non-default ports.

It is possible for the user to specify an alternate data port by use of the PORT command. The user may want a file dumped on a TAC line printer or retrieved from a third party host. In the latter case, the user-PI sets up control connections with both server-PI's. One server is then told (by an FTP command) to "listen" for a connection which the other will initiate. The user-PI sends one server-PI a PORT command indicating the data port of the other. Finally, both are sent the appropriate transfer commands. The exact sequence of commands and replies sent between the user-controller and the servers is defined in the Section on FTP Replies.

In general, it is the server's responsibility to maintain the data connection--to initiate it and to close it. The exception to this is when the user-DTP is sending the data in a transfer mode that requires the connection to be closed to indicate EOF. The server **must** close the data connection under the following conditions:

1. The server has completed sending data in a transfer mode that requires a close to indicate EOF.
2. The server receives an ABORT command from the user.
3. The port specification is changed by a command from the user.
4. The control connection is closed legally or otherwise.
5. An irrecoverable error condition occurs.

Otherwise the close is a server option, the exercise of which the server must indicate to the user-process by either a 250 or 226 reply only.

RFC-959 File Transfer Protocol (FTP) -- Data Transfer Functions

Data Connection Management

Default Data Connection Ports: All FTP implementations must support use of the default data connection ports, and only the User-PI may initiate the use of non-default ports.

Negotiating Non-Default Data Ports: The User-PI may specify a non-default user side data port with the PORT command. The User-PI may request the server side to identify a non-default server side data port with the PASV command. Since a connection is defined by the pair of addresses, either of these actions is enough to get a different data connection, still it is permitted to do both commands to use new ports on both ends of the data connection.

Reuse of the Data Connection: When using the stream mode of data transfer the end of the file must be indicated by closing the connection. This causes a problem if multiple files are to be transfered in the session, due to need for TCP to hold the connection record for a time out period to guarantee the reliable communication. Thus the connection can not be reopened at once.

There are two solutions to this problem. The first is to negotiate a non-default port. The second is to use another transfer mode.

A comment on transfer modes. The stream transfer mode is inherently unreliable, since one can not determine if the connection closed prematurely or not. The other transfer modes (Block, Compressed) do not close the connection to indicate the end of file. They have enough FTP encoding that the data connection can be parsed to determine the end of the file. Thus using these modes one can leave the data connection open for multiple file transfers.

A User-FTP that uses STREAM mode **should** send a PORT command to assign a non-default data port before each transfer command is issued.

> **Discussion**
> This is required because of the long delay after a TCP connection is closed until its socket pair can be reused, to allow multiple transfers duringa single FTP session. Sending a port command can avoided if a transfer mode other than stream is used, by leaving the data transfer connection open between transfers.

Transmission Modes

The next consideration in transferring data is choosing the appropriate transmission mode. There are three modes: one which formats the data and allows for restart procedures; one which also compresses the data for efficient transfer; and one which passes the data with little or no processing. In this last case the mode interacts with the structure attribute to determine the type of processing. In the compressed mode, the representation type determines the filler byte.

All data transfers must be completed with an end-of-file (EOF) which may be explicitly stated or implied by the closing of the data connection. For files with record structure, all the end-of-record markers (EOR) are explicit, including the final one. For files transmitted in page structure a "last-page" page type is used.

Note: In the rest of this section, byte means "transfer byte" except where explicitly stated otherwise.

For the purpose of standardized transfer, the sending host will translate its internal end of line or end of record denotation into the representation prescribed by the transfer mode and file structure, and the receiving host will perform the inverse translation to its internal denotation. An IBM Mainframe record count field may not be recognized at another host, so the end-of-record information may be transferred as a two byte control code in Stream mode or as a flagged bit in a Block or Compressed mode descriptor. End-of-line in an ASCII or EBCDIC file with no record structure should be indicated by <CRLF> or <NL>, respectively. Since these transformations imply extra work for some systems, identical systems transferring non-record structured text files might wish to use a binary representation and stream mode for the transfer.

Stream Mode
Block Mode
Compressed Mode

Stream Mode

The data is transmitted as a stream of bytes. There is no restriction on the representation type used; record structures are allowed.

In a record structured file EOR and EOF will each be indicated by a two-byte control code. The first byte of the control code will be all ones, the escape character. The second byte will have the low order bit on and zeros elsewhere for EOR and the second low order bit on for EOF; that is, the byte will have value 1 for EOR and value 2 for EOF. EOR and EOF may be indicated together on the last byte transmitted by turning both low order bits on (i.e., the value 3). If a byte of all ones was intended to be sent as data, it should be repeated in the second byte of the control code.

If the structure is a file structure, the EOF is indicated by the sending host closing the data connection and all bytes are data bytes.

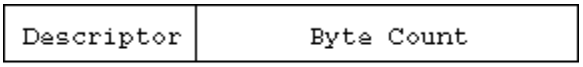
RFC-959 File Transfer Protocol (FTP) -- Transmission Modes

Block Mode

The file is transmitted as a series of data blocks preceded by one or more header bytes. The header bytes contain a count field, and descriptor code. The count field indicates the total length of the data block in bytes, thus marking the beginning of the next data block (there are no filler bits). The descriptor code defines: last block in the file (EOF) last block in the record (EOR), restart marker or suspect data (i.e., the data being transferred is suspected of errors and is not reliable). This last code is **not** intended for error control within FTP. It is motivated by the desire of sites exchanging certain types of data (e.g., seismic or weather data) to send and receive all the data despite local errors (such as "magnetic tape read errors"), but to indicate in the transmission that certain portions are suspect). Record structures are allowed in this mode, and any representation type may be used.

The header consists of the three bytes. Of the 24 bits of header information, the 16 low order bits shall represent byte count, and the 8 high order bits shall represent descriptor codes as shown below.

Block Header



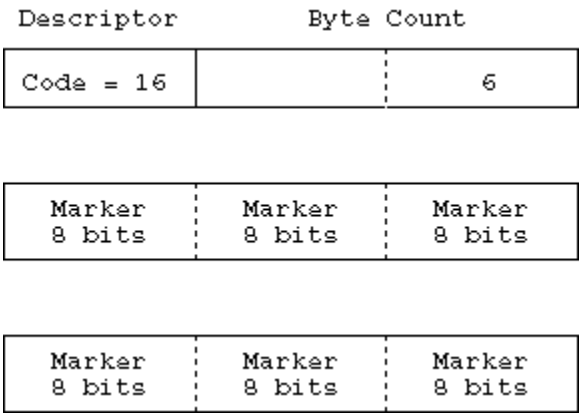
The descriptor codes are indicated by bit flags in the descriptor byte. Four codes have been assigned, where each code number is the decimal value of the corresponding bit in the byte.

Code	Meaning
128	End of data block is EOR
64	End of data block is EOF
32	Suspected errors in data block
16	Data block is a restart marker

With this encoding, more than one descriptor coded condition may exist for a particular block. As many bits as necessary may be flagged.

The restart marker is embedded in the data stream as an integral number of 8-bit bytes representing printable characters in the language being used over the control connection (e.g., default--NVT-ASCII). <SP> (Space, in the appropriate language) must not be used **within** a restart marker.

For example, to transmit a six-character marker, the following would be sent:

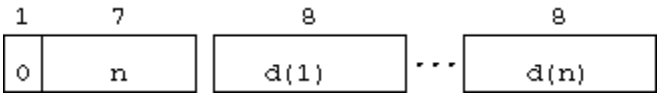

Descriptor
Byte Count
Code = 16
6
Marker
8 bits
Marker
8 bits
Marker
8 bits
Marker
8 bits
Marker
8 bits
Marker
8 bits

RFC-959 File Transfer Protocol (FTP) -- Transmission Modes

Compressed Mode

There are three kinds of information to be sent: regular data, sent in a byte string; compressed data, consisting of replications or filler; and control information, sent in a two-byte escape sequence. If n>0 bytes (up to 127) of regular data are sent, these n bytes are preceded by a byte with the left-most bit set to 0 and the right-most 7 bits containing the number n.

Byte string:



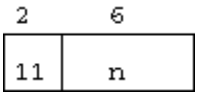
String of n data bytes d(1),..., d(n)
Count n must be positive.

To compress a string of n replications of the data byte d, the following 2 bytes are sent:

Replicated Byte:

A string of n filler bytes can be compressed into a single byte, where the filler byte varies with the representation type. If the type is ASCII or EBCDIC the filler byte is <SP> (Space, ASCII code 32, EBCDIC code 64). If the type is Image or Local byte the filler is a zero byte.

Filler String:



The escape sequence is a double byte, the first of which is the escape byte (all zeros) and the second of which contains descriptor codes as defined in Block mode. The descriptor codes have the same meaning as in Block mode and apply to the succeeding string of bytes.

Compressed mode is useful for obtaining increased bandwidth on very large network transmissions at a little extra CPU cost. It can be most effectively used to reduce the size of printer files such as those generated by RJE hosts.

Error Recovery And Restart

There is no provision for detecting bits lost or scrambled in data transfer; this level of error control is handled by the TCP. However, a restart procedure is provided to protect users from gross system failures (including failures of a host, an FTP-process, or the underlying network).

The restart procedure is defined only for the block and compressed modes of data transfer. It requires the sender of data to insert a special marker code in the data stream with some marker information. The marker information has meaning only to the sender, but must consist of printable characters in the default or negotiated language of the control connection (ASCII or EBCDIC). The marker could represent a bit-count, a record-count, or any other information by which a system may identify a data checkpoint. The receiver of data, if it implements the restart procedure, would then mark the corresponding position of this marker in the receiving system, and return this information to the user.

In the event of a system failure, the user can restart the data transfer by identifying the marker point with the FTP restart procedure. The following example illustrates the use of the restart procedure.

The sender of the data inserts an appropriate marker block in the data stream at a convenient point. The receiving host marks the corresponding data point in its file system and conveys the last known sender and receiver marker information to the user, either directly or over the control connection in a 110 reply (depending on who is the sender). In the event of a system failure, the user or controller process restarts the server at the last server marker by sending a restart command with server's marker code as its argument. The restart command is transmitted over the control connection and is immediately followed by the command (such as RETR, STOR or LIST) which was being executed when the system failure occurred.

When an FTP that implements restart receives a Restart Marker in the data stream, it **should** force the data to that point to be written to stable storage beforeencoding the corresponding position rrrr. An FTP sending Restart Markers **must not** assume that 110 replies will be returned synchronously with the data, i.e., it must not await a 110 reply before sending more data.

Two new reply codes are hereby defined for errors encountered in restarting a transfer:

554 Requested action not taken: invalid REST parameter.

A 554 reply may result from a FTP service command that follows a REST command. The reply indicates that the existing file at the Server-FTP cannot be repositioned as specified in the REST.

555 Requested action not taken: type or stru mismatch.

A 555 reply may result from an APPE command or from any FTP service command following a REST command. The reply indicates that there is some mismatch between the current transfer parameters (type and stru) and the attributes of the existing file.

Restart Markers mark a place in the data stream, but the receiver may be performing some transformation on the data as it is stored into stable storage. In general, the receiver's encoding must include any state information necessary to restart this transformation at any point of the FTP data stream. For example, in TYPE A transfers, some receiver hosts transform CR LF sequences into a single LF character on disk. If a Restart Marker happens

to fall between CR and LF, the receiver must encode in rrrr that the transfer must be restarted in a "CR has been seen and discarded" state.

Note that the Restart Marker is required to be encoded as a string of printable ASCII characters, regardless of the type of the data.

The discussion above mentions that restart information is to be returned "to the user". This should not be taken literally. In general, the User-FTP should save the restart information (ssss,rrrr) in stable storage, e.g., append it to a restart control file. An empty restart control file should be created when the transfer first starts and deleted automatically when the transfer completes successfully. It is suggested that this file have a name derived in an easily-identifiable manner from the name of the file being transferred and the remote host name; this is analogous to the means used by many text editors for naming "backup" files.

There are three cases for FTP restart:

(1) User-to-Server Transfer

The User-FTP puts Restart Markers <ssss> at convenient places in the data stream. When the Server-FTP receives a Marker, it writes all prior data to disk, encodes its file system position and transformation state as rrrr, and returns a "110 MARK ssss = rrrr" reply over the control connection. The User-FTP appends the pair (ssss,rrrr) to its restart control file.

To restart the transfer, the User-FTP fetches the last (ssss,rrrr) pair from the restart control file, repositions its local file system and transformation state using ssss, and sends the command "REST rrrr" to the Server-FTP.

(2) Server-to-User Transfer

The Server-FTP puts Restart Markers <ssss> at convenient places in the data stream. When the User-FTP receives a Marker, it writes all prior data to disk, encodes its file system position and transformation state as rrrr, and appends the pair (rrrr,ssss) to its restart control file.

To restart the transfer, the User-FTP fetches the last (rrrr,ssss) pair from the restart control file, repositions its local file system and transformation state using rrrr, and sends the command "REST ssss" to the Server-FTP.

(3) Server-to-Server ("Third-Party") Transfer

The sending Server-FTP puts Restart Markers <ssss> at convenient places in the data stream. When it receives a Marker, the receiving Server-FTP writes all prior data to disk, encodes its file system position and transformation state as rrrr, and sends a "110 MARK ssss = rrrr" reply over the control connection to theUser. The User-FTP appends the pair (ssss,rrrr) to its restart control file.

To restart the transfer, the User-FTP fetches the last (ssss,rrrr) pair from the restart control file, sends "REST ssss" to the sending Server-FTP, and sends "REST rrrr" to the receiving Server-FTP.

File Transfer Functions

The communication channel from the user-PI to the server-PI is established as a TCP connection from the user to the standard server port. The user protocol interpreter is responsible for sending FTP commands and interpreting the replies received; the server-PI interprets commands, sends replies and directs its DTP to set up the data connection and transfer the data. If the second party to the data transfer (the passive transfer process) is the user-DTP, then it is governed through the internal protocol of the user-FTP host; if it is a second server-DTP, then it is governed by its PI on command from the user-PI. The FTP replies are discussed in the next section. In the description of a few of the commands in this section, it is helpful to be explicit about the possible replies.

FTP Commands
FTP Replies

RFC-959 File Transfer Protocol (FTP)

FTP Commands

The File Transfer Protocol follows the specifications of the Telnet protocol for all communications over the control connection. Since the language used for Telnet communication may be a negotiated option, all references in the next two sections will be to the "Telnet language" and the corresponding "Telnet end-of-line code". Currently, one may take these to mean NVT-ASCII and <CRLF>. No other specifications of the Telnet protocol will be cited.

FTP commands are "Telnet strings" terminated by the "Telnet end of line code". The command codes themselves are alphabetic characters terminated by the character <SP> (Space) if parameters follow and Telnet-EOL otherwise. The command codes and the semantics of commands are described in this section; the detailed syntax of commands is specified in Commands, the reply sequences are discussed in Sequencing of Commands and Replies, and scenarios illustrating the use of commands are provided in Typical FTP Scenarios.

FTP commands may be partitioned as those specifying access-control identifiers, data transfer parameters, or FTP service requests. Certain commands (such as ABOR, STAT, QUIT) may be sent over the control connection while a data transfer is in progress. Some servers may not be able to monitor the control and data connections simultaneously, in which case some special action will be necessary to get the server's attention. The following ordered format is tentatively recommended:

1. User system inserts the Telnet "Interrupt Process" (IP) signal in the Telnet stream.
2. User system sends the Telnet "Synch" signal.
3. User system inserts the command (e.g., ABOR) in the Telnet stream.
4. Server PI, after receiving "IP", scans the Telnet stream for **exactly one** FTP command.

(For other servers this may not be necessary but the actions listed above should have no unusual effect.)

Implementors **must not** assume any correspondence between READ boundaries on the control connection and the Telnet EOL sequences (CR LF).

Discussion
Thus, a server-FTP (or User-FTP) must continue reading characters from the control connection until a complete Telnet EOL sequence is encountered, before processing the command (or response, respectively). Conversely, a single READ from the control connection may include more than one FTP command.

Access Control Commands

The following commands specify access control identifiers (command codes are shown in parentheses).

User Name
Password
Account
Change Working Directory
Change to Parent Directory
Structure Mount
Reinitialize
Logout

RFC-959 File Transfer Protocol (FTP) -- Access Control Commands

USER NAME (USER)

The argument field is a Telnet string identifying the user. The user identification is that which is required by the server for access to its file system. This command will normally be the first command transmitted by the user after the control connections are made (some servers may require this). Additional identification information in the form of a password and/or an account command may also be required by some servers. Servers may allow a new USER command to be entered at any point in order to change the access control and/or accounting information. This has the effect of flushing any user, password, and account information already supplied and beginning the login sequence again. All transfer parameters are unchanged and any file transfer in progress is completed under the old access control parameters.

RFC-959 File Transfer Protocol (FTP) -- Access Control Commands

PASSWORD (PASS)

The argument field is a Telnet string specifying the user's password. This command must be immediately preceded by the user name command, and, for some sites, completes the user's identification for access control. Since password information is quite sensitive, it is desirable in general to "mask" it or suppress typeout. It appears that the server has no foolproof way to achieve this. It is therefore the responsibility of the user-FTP process to hide the sensitive password information.

RFC-959 File Transfer Protocol (FTP) -- Access Control Commands

ACCOUNT (ACCT)

The argument field is a Telnet string identifying the user's account. The command is not necessarily related to the USER command, as some sites may require an account for login and others only for specific access, such as storing files. In the latter case the command may arrive at any time.

There are reply codes to differentiate these cases for the automation: when account information is required for login, the response to a successful PASSword command is reply code 332. On the other hand, if account information is NOT required for login, the reply to a successful PASSword command is 230; and if the account information is needed for a command issued later in the dialogue, the server should return a 332 or 532 reply depending on whether it stores (pending receipt of the ACCounT command) or discards the command, respectively.

RFC-959 File Transfer Protocol (FTP) -- Access Control Commands

CHANGE WORKING DIRECTORY (CWD)

This command allows the user to work with a different directory or dataset for file storage or retrieval without altering his login or accounting information. Transfer parameters are similarly unchanged. The argument is a pathname specifying a directory or other system dependent file group designator.

CHANGE TO PARENT DIRECTORY (CDUP)

This command is a special case of CWD, and is included to simplify the implementation of programs for transferring directory trees between operating systems having different syntaxes for naming the parent directory. The reply codes shall be identical to the reply codes of CWD. See Appendix II for further details.

RFC-959 File Transfer Protocol (FTP) -- Access Control Commands

STRUCTURE MOUNT (SMNT)

This command allows the user to mount a different file system data structure without altering his login or accounting information. Transfer parameters are similarly unchanged. The argument is a pathname specifying a directory or other system dependent file group designator.

RFC-959 File Transfer Protocol (FTP) -- Access Control Commands

REINITIALIZE (REIN)

This command terminates a USER, flushing all I/O and account information, except to allow any transfer in progress to be completed. All parameters are reset to the default settings and the control connection is left open. This is identical to the state in which a user finds himself immediately after the control connection is opened. A USER command may be expected to follow.

RFC-959 File Transfer Protocol (FTP) -- Access Control Commands

LOGOUT (QUIT)

This command terminates a USER and if file transfer is not in progress, the server closes the control connection. If file transfer is in progress, the connection will remain open for result response and the server will then close it. If the user-process is transferring files for several USERs but does not wish to close and then reopen connections for each, then the REIN command should be used instead of QUIT.

An unexpected close on the control connection will cause the server to take the effective action of an abort (ABOR) and a logout (QUIT).

Transfer Parameter Commands

All data transfer parameters have default values, and the commands specifying data transfer parameters are required only if the default parameter values are to be changed. The default value is the last specified value, or if no value has been specified, the standard default value is as stated here. This implies that the server must "remember" the applicable default values. The commands may be in any order except that they must precede the FTP service request. The following commands specify data transfer parameters:

Data Port
Passive
Representation Type
FIle Structure
Transfer Mode

RFC-959 File Transfer Protocol (FTP) -- Transfer Parameter Commands

DATA PORT (PORT)

The argument is a HOST-PORT specification for the data port to be used in data connection. There are defaults for both the user and server data ports, and under normal circumstances this command and its reply are not needed. If this command is used, the argument is the concatenation of a 32-bit internet host address and a 16-bit TCP port address. This address information is broken into 8-bit fields and the value of each field is transmitted as a decimal number (in character string representation). The fields are separated by commas. A port command would be:

PORT h1,h2,h3,h4,p1,p2

where h1 is the high order 8 bits of the internet host address.

RFC-959 File Transfer Protocol (FTP) -- Transfer Parameter Commands

PASSIVE (PASV)

> This command requests the server-DTP to "listen" on a data port (which is not its default data port) and to wait for a connection rather than initiate one upon receipt of a transfer command. The response to this command includes the host and port address this server is listening on.

A server-FTP **must** implement the PASV command.

If multiple third-party transfers are to be executed during the same session, a new PASV command **must** be issued before each transfer command, to obtain a unique port pair.

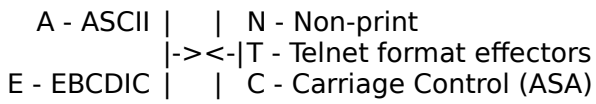
> **Implementation**
> The format of the 227 reply to a PASV command is not well standardized. In particular, an FTP client cannot assume that the parentheses shown on page 40 of RFC-959 will be present (and in fact, Figure 3 on page 43 omits them). Therefore, a User-FTP program that interprets the PASV reply must scan the reply for the first digit of the host and port numbers.
>
> Note that the host number h1,h2,h3,h4 is the IP address of the server host that is sending the reply, and that p1,p2 is a non-default data transfer port that PASV has assigned.

RFC-959 File Transfer Protocol (FTP) -- Transfer Parameter Commands

REPRESENTATION TYPE (TYPE)

The argument specifies the representation type as described in the Section on Data Representation and Storage. Several types take a second parameter. The first parameter is denoted by a single Telnet character, as is the second Format parameter for ASCII and EBCDIC; the second parameter for local byte is a decimal integer to indicate Bytesize. The parameters are separated by a <SP> (Space, ASCII code 32).

The following codes are assigned for type:



I - Image

L <byte size> - Local byte Byte size

The default representation type is ASCII Non-print. If the Format parameter is changed, and later just the first argument is changed, Format then returns to the Non-print default.

RFC-959 File Transfer Protocol (FTP) -- Transfer Parameter Commands

FILE STRUCTURE (STRU)

The argument is a single Telnet character code specifying file structure described in the Section on Data Representation and Storage.

The following codes are assigned for structure:

- F - File (no record structure)
- R - Record structure
- P - Page structure

The default structure is File.

RFC-959 File Transfer Protocol (FTP) -- Transfer Parameter Commands

TRANSFER MODE (MODE)

The argument is a single Telnet character code specifying the data transfer modes described in the Section on Transmission Modes.

The following codes are assigned for transfer modes:

S - Stream
B - Block
C - Compressed

The default transfer mode is Stream.

RFC-959 File Transfer Protocol (FTP) -- Commands

FTP Service Commands

The FTP service commands define the file transfer or the file system function requested by the user. The argument of an FTP service command will normally be a pathname. The syntax of pathnames must conform to server site conventions (with standard defaults applicable), and the language conventions of the control connection. The suggested default handling is to use the last specified device, directory or file name, or the standard default defined for local users. The commands may be in any order except that a "rename from" command must be followed by a "rename to" command and the restart command must be followed by the interrupted service command (e.g., STOR or RETR). The data, when transferred in response to FTP service commands, shall always be sent over the data connection, except for certain informative replies. The following commands specify FTP service requests:

Retrieve	**Store**
Store Unique	**Append**
Allocate	**Restart**
Rename From	**Rename To**
Abort	**Delete**
Remove Directory	**Make Directory**
Print Working Directory	**List**
Name List	**Site Parameters**
System	**Status**
Help	**NoOp**
Quote	

RFC-959 File Transfer Protocol (FTP) -- Service Commands

RETRIEVE (RETR)

This command causes the server-DTP to transfer a copy of the file, specified in the pathname, to the server- or user-DTP at the other end of the data connection. The status and contents of the file at the server site shall be unaffected.

RFC-959 File Transfer Protocol (FTP) -- Service Commands

STORE (STOR)

This command causes the server-DTP to accept the data transferred via the data connection and to store the data as a file at the server site. If the file specified in the pathname exists at the server site, then its contents shall be replaced by the data being transferred. A new file is created at the server site if the file specified in the pathname does not already exist.

STORE UNIQUE (STOU)

This command behaves like STOR except that the resultant file is to be created in the current directory under a name unique to that directory. When it receives an STOU command, a Server-FTP **must** return the actual file name in the "125 Transfer Starting" or the "150 Opening Data Connection" message that precedes the transfer. The exact format of thesemessages is hereby defined to be as follows:

125 FILE: pppp
150 FILE: pppp

where pppp represents the unique pathname of the file that will be written.

RFC-959 File Transfer Protocol (FTP) -- Service Commands

APPEND (with create) (APPE)

This command causes the server-DTP to accept the data transferred via the data connection and to store the data in a file at the server site. If the file specified in the pathname exists at the server site, then the data shall be appended to that file; otherwise the file specified in the pathname shall be created at the server site.

RFC-959 File Transfer Protocol (FTP) -- Service Commands

ALLOCATE (ALLO)

This command may be required by some servers to reserve sufficient storage to accommodate the new file to be transferred. The argument shall be a decimal integer representing the number of bytes (using the logical byte size) of storage to be reserved for the file. For files sent with record or page structure a maximum record or page size (in logical bytes) might also be necessary; this is indicated by a decimal integer in a second argument field of the command. This second argument is optional, but when present should be separated from the first by the three Telnet characters <SP> R <SP>. This command shall be followed by a STORe or APPEnd command. The ALLO command should be treated as a NOOP (no operation) by those servers which do not require that the maximum size of the file be declared beforehand, and those servers interested in only the maximum record or page size should accept a dummy value in the first argument and ignore it.

RESTART (REST)

The argument field represents the server marker at which file transfer is to be restarted. This command does not cause file transfer but skips over the file to the specified data checkpoint. This command shall be immediately followed by the appropriate FTP service command which shall cause file transfer to resume.

RENAME FROM (RNFR)

This command specifies the old pathname of the file which is to be renamed. This command must be immediately followed by a "rename to" command specifying the new file pathname.

RENAME TO (RNTO)

This command specifies the new pathname of the file specified in the immediately preceding "rename from" command. Together the two commands cause a file to be renamed.

RFC-959 File Transfer Protocol (FTP) -- Service Commands

ABORT (ABOR)

This command tells the server to abort the previous FTP service command and any associated transfer of data. The abort command may require "special action", as discussed in the Section on FTP Commands, to force recognition by the server. No action is to be taken if the previous command has been completed (including data transfer). The control connection is not to be closed by the server, but the data connection must be closed.

There are two cases for the server upon receipt of this command: (1) the FTP service command was already completed, or (2) the FTP service command is still in progress.

In the first case, the server closes the data connection (if it is open) and responds with a 226 reply, indicating that the abort command was successfully processed.

In the second case, the server aborts the FTP service in progress and closes the data connection, returning a 426 reply to indicate that the service request terminated abnormally. The server then sends a 226 reply, indicating that the abort command was successfully processed.

RFC-959 File Transfer Protocol (FTP) -- Service Commands

DELETE (DELE)

This command causes the file specified in the pathname to be deleted at the server site. If an extra level of protection is desired (such as the query, "Do you really wish to delete?"), it should be provided by the user-FTP process.

REMOVE DIRECTORY (RMD)

This command causes the directory specified in the pathname to be removed as a directory (if the pathname is absolute) or as a subdirectory of the current working directory (if the pathname is relative). See Appendix II.

MAKE DIRECTORY (MKD)

This command causes the directory specified in the pathname to be created as a directory (if the pathname is absolute) or as a subdirectory of the current working directory (if the pathname is relative). See Appendix II.

PRINT WORKING DIRECTORY (PWD)

This command causes the name of the current working directory to be returned in the reply. See Appendix II.

RFC-959 File Transfer Protocol (FTP) -- Service Commands

LIST (LIST)

> This command causes a list to be sent from the server to the passive DTP. If the pathname specifies a directory or other group of files, the server should transfer a list of files in the specified directory. If the pathname specifies a file then the server should send current information on the file. A null argument implies the user's current working or default directory. The data transfer is over the data connection in type ASCII or type EBCDIC. (The user must ensure that the TYPE is appropriately ASCII or EBCDIC). Since the information on a file may vary widely from system to system, this information may be hard to use automatically in a program, but may be quite useful to a human user.

The data returned by a LIST or NLST command **should** use an implied TYPE AN, unless the current type is EBCDIC, in which case an implied TYPE EN **should** be used.

> **Discussion**
> Many FTP clients support macro-commands that will get or put files matching a wildcard specification, using NLST to obtain a list of pathnames. The expansion of "multiple-put" is local to the client, but "multiple- get" requires cooperation by the server.
>
> The implied type for LIST and NLST is designed to provide compatibility with existing User-FTPs, and in particular with multiple-get commands.

RFC-959 File Transfer Protocol (FTP) -- Service Commands

NAME LIST (NLST)

> This command causes a directory listing to be sent from server to user site. The pathname should specify a directory or other system-specific file group descriptor; a null argument implies the current directory. The server will return a stream of names of files and no other information. The data will be transferred in ASCII or EBCDIC type over the data connection as valid pathname strings separated by <CRLF> or <NL>. (Again the user must ensure that the TYPE is correct.) This command is intended to return information that can be used by a program to further process the files automatically. For example, in the implementation of a "multiple get" function.

The data returned by an NLST command **must** contain only a simple list of legal pathnames, such that the server can use them directly as the arguments of subsequent data transfer commands for the individual files.

The data returned by a LIST or NLST command **should** use an implied TYPE AN, unless the current type is EBCDIC, in which case an implied TYPE EN **should** be used.

> **Discussion**
> Many FTP clients support macro-commands that will get or put files matching a wildcard specification, using NLST to obtain a list of pathnames. The expansion of "multiple-put" is local to the client, but "multiple- get" requires cooperation by the server.
>
> The implied type for LIST and NLST is designed to provide compatibility with existing User-FTPs, and in particular with multiple-get commands.

RFC-959 File Transfer Protocol (FTP) -- Service Commands

SITE PARAMETERS (SITE)

> This command is used by the server to provide services specific to his system that are essential to file transfer but not sufficiently universal to be included as commands in the protocol. The nature of these services and the specification of their syntax can be stated in a reply to the HELP SITE command.

A Server-FTP **should** use the SITE command for non-standard features, rather than invent new private commands or unstandardized extensions to existing commands.

SYSTEM (SYST)

This command is used to find out the type of operating system at the server. The reply shall have as its first word one of the system names listed in the current version of the Assigned Numbers document [4].

RFC-959 File Transfer Protocol (FTP) -- Service Commands

STATUS (STAT)

This command shall cause a status response to be sent over the control connection in the form of a reply. The command may be sent during a file transfer (along with the Telnet IP and Synch signals--see the Section on FTP Commands) in which case the server will respond with the status of the operation in progress, or it may be sent between file transfers. In the latter case, the command may have an argument field. If the argument is a pathname, the command is analogous to the "list" command except that data shall be transferred over the control connection. If a partial pathname is given, the server may respond with a list of file names or attributes associated with that specification. If no argument is given, the server should return general status information about the server FTP process. This should include current values of all transfer parameters and the status of connections.

RFC-959 File Transfer Protocol (FTP) -- Service Commands

HELP (HELP)

This command shall cause the server to send helpful information regarding its implementation status over the control connection to the user. The command may take an argument (e.g., any command name) and return more specific information as a response. The reply is type 211 or 214. It is suggested that HELP be allowed before entering a USER command. The server may use this reply to specify site-dependent parameters, e.g., in response to HELP SITE.

RFC-959 File Transfer Protocol (FTP) -- Service Commands

NOOP (NOOP)

This command does not affect any parameters or previously entered commands. It specifies no action other than that the server send an OK reply.

RFC-959 File Transfer Protocol (FTP) -- Service Commands

Quote

A User-FTP program **must** implement a "QUOTE" command that will pass an arbitrary character string to the server and display all resulting response messages to the user.

To make the "QUOTE" command useful, a User-FTP **should** send transfer control commands to the server as the user enters them, rather than saving all the commandsand sending them to the server only when a data transfer is started.

Discussion
The "QUOTE" command is essential to allow the user to access servers that require system-specific commands (e.g., SITE or ALLO), or to invoke new or optional features that are not implemented by the User-FTP. For example, "QUOTE" may be used to specify "TYPE A T" to send a print file to hosts that require the distinction, even if the User-FTP does not recognize that TYPE.

RFC-959 File Transfer Protocol (FTP)

FTP Replies

Replies to File Transfer Protocol commands are devised to ensure the synchronization of requests and actions in the process of file transfer, and to guarantee that the user process always knows the state of the Server. Every command must generate at least one reply, although there may be more than one; in the latter case, the multiple replies must be easily distinguished. In addition, some commands occur in sequential groups, such as USER, PASS and ACCT, or RNFR and RNTO. The replies show the existence of an intermediate state if all preceding commands have been successful. A failure at any point in the sequence necessitates the repetition of the entire sequence from the beginning.

> The details of the command-reply sequence are made explicit in a set of state diagrams below.

An FTP reply consists of a three digit number (transmitted as three alphanumeric characters) followed by some text. The number is intended for use by automata to determine what state to enter next; the text is intended for the human user. It is intended that the three digits contain enough encoded information that the user-process (the User-PI) will not need to examine the text and may either discard it or pass it on to the user, as appropriate. In particular, the text may be server-dependent, so there are likely to be varying texts for each reply code.

A reply is defined to contain the 3-digit code, followed by Space <SP>, followed by one line of text (where some maximum line length has been specified), and terminated by the Telnet end-of-line code. There will be cases however, where the text is longer than a single line. In these cases the complete text must be bracketed so the User-process knows when it may stop reading the reply (i.e. stop processing input on the control connection) and go do other things. This requires a special format on the first line to indicate that more than one line is coming, and another on the last line to designate it as the last. At least one of these must contain the appropriate reply code to indicate the state of the transaction. To satisfy all factions, it was decided that both the first and last line codes should be the same.

> Thus the format for multi-line replies is that the first line will begin with the exact required reply code, followed immediately by a Hyphen, "-" (also known as Minus), followed by text. The last line will begin with the same code, followed immediately by Space <SP>, optionally some text, and the Telnet end-of-line code.
>
> For example:
>
> 123-First line
> Second line
> 234 A line beginning with numbers
> 123 The last line
>
> The user-process then simply needs to search for the second occurrence of the same reply code, followed by <SP> (Space), at the beginning of a line, and ignore all intermediary lines. If an intermediary line begins with a 3-digit number, the Server must pad the front to avoid confusion.
>
> This scheme allows standard system routines to be used for reply information

(such as for the STAT reply), with "artificial" first and last lines tacked on. In rare cases where these routines are able to generate three digits and a Space at the beginning of any line, the beginning of each text line should be offset by some neutral text, like Space.

This scheme assumes that multi-line replies may not be nested.

RFC-959 File Transfer Protocol (FTP) -- Reply Codes

Formal Specification

The three digits of the reply each have a special significance. This is intended to allow a range of very simple to very sophisticated responses by the user-process. The first digit denotes whether the response is good, bad or incomplete. (Referring to the state diagram), an unsophisticated user-process will be able to determine its next action (proceed as planned, redo, retrench, etc.) by simply examining this first digit. A user-process that wants to know approximately what kind of error occurred (e.g. file system error, command syntax error) may examine the second digit, reserving the third digit for the finest gradation of information (e.g., RNTO command without a preceding RNFR).

There are five values for the first digit of the reply code:

1yz Positive Preliminary reply

The requested action is being initiated; expect another reply before proceeding with a new command. (The user-process sending another command before the completion reply would be in violation of protocol; but server-FTP processes should queue any commands that arrive while a preceding command is in progress.) This type of reply can be used to indicate that the command was accepted and the user-process may now pay attention to the data connections, for implementations where simultaneous monitoring is difficult. The server-FTP process may send at most, one 1yz reply per command.

2yz Positive Completion reply

The requested action has been successfully completed. A new request may be initiated.

3yz Positive Intermediate reply

The command has been accepted, but the requested action is being held in abeyance, pending receipt of further information. The user should send another command specifying this information. This reply is used in command sequence groups.

4yz Transient Negative Completion reply

The command was not accepted and the requested action did not take place, but the error condition is temporary and the action may be requested again. The user should return to the beginning of the command sequence, if any. It is difficult to assign a meaning to "transient", particularly when two distinct sites (Server- and User-processes) have to agree on the interpretation. Each reply in the 4yz category might have a slightly different time value, but the intent is that the user-process is encouraged to try again. A rule of thumb in determining if a reply fits into the 4yz or the 5yz (Permanent Negative) category is that replies are 4yz if the commands can be repeated without any change in command form or in properties of the User or Server (e.g., the command is spelled the same with the same arguments used; the user does not change his file access or user name; the server does not put up a new implementation.)

5yz Permanent Negative Completion reply

The command was not accepted and the requested action did not take place. The User-process is discouraged from repeating the exact request (in the

same sequence). Even some "permanent" error conditions can be corrected, so the human user may want to direct his User-process to reinitiate the command sequence by direct action at some point in the future (e.g., after the spelling has been changed, or the user has altered his directory status.)

The following function groupings are encoded in the second digit:

x0zSyntax - These replies refer to syntax errors, syntactically correct commands that don't fit any functional category, unimplemented or superfluous commands.

x1zInformation - These are replies to requests for information, such as status or help.

x2zConnections - Replies referring to the control and data connections.

x3zAuthentication and accounting - Replies for the login process and accounting procedures.

x4zUnspecified as yet.

x5zFile system - These replies indicate the status of the Server file system vis-a-vis the requested transfer or other file system action.

The third digit gives a finer gradation of meaning in each of the function categories, specified by the second digit. The list of replies below will illustrate this. Note that the text associated with each reply is recommended, rather than mandatory, and may even change according to the command with which it is associated. The reply codes, on the other hand, must strictly follow the specifications in the last section; that is, Server implementations should not invent new codes for situations that are only slightly different from the ones described here, but rather should adapt codes already defined.

A command such as TYPE or ALLO whose successful execution does not offer the user-process any new information will cause a 200 reply to be returned. If the command is not implemented by a particular Server-FTP process because it has no relevance to that computer system, for example ALLO at a TOPS20 site, a Positive Completion reply is still desired so that the simple User-process knows it can proceed with its course of action. A 202 reply is used in this case with, for example, the reply text: "No storage allocation necessary." If, on the other hand, the command requests a non-site-specific action and is unimplemented, the response is 502. A refinement of that is the 504 reply for a command that is implemented, but that requests an unimplemented parameter.

RFC-959 File Transfer Protocol (FTP) -- Reply Codes

General Notes

A Server-FTP **must** send only correctly formatted replies on the control connection. Note that this specification (unlike earlier versions of the FTP spec) contains no provision for a "spontaneous" reply message.

A Server-FTP **should** use the reply codes defined in Formal Specification whenever they apply. However, a server-FTP MAY use a different reply code when needed, as long as the general rules of the specification are followed. When the implementor has a choice between a 4xx and 5xx reply code, a Server-FTP **should** send a 4xx (temporary failure) code when there is any reasonable possibility that a failed FTP will succeed a few hours later.

A User-FTP **should** generally use only the highest-order digit of a 3-digit reply code for making a procedural decision, to prevent difficulties when a Server-FTP uses non-standard reply codes.

A User-FTP **must** be able to handle multi-line replies. If the implementation imposes a limit on the number of lines and if this limit is exceeded, the User-FTP**must** recover, e.g., by ignoring the excess lines until the end of the multi-line reply is reached.

A User-FTP **should not** interpret a 421 reply code ("Service not available, closing control connection") specially, but **should** detect closing of the control connection by the server.

> **Discussion**
> Server implementations that fail to strictly follow the reply rules often cause FTP user programs to hang. Note that RFC-959 resolved ambiguities in the reply rules found in earlier FTP specifications and must be followed.
>
> It is important to choose FTP reply codes that properly distinguish between temporary and permanent failures, to allow the successful use of file transfer client daemons. These programs depend on the reply codes to decide whether or not to retry a failed transfer; using a permanent failure code (5xx) for a temporaryerror will cause these programs to give up unnecessarily.
>
> When the meaning of a reply matches exactly the text shown in RFC-959, uniformity will be enhanced by using the RFC-959 text verbatim. However, a Server-FTP implementor is encouraged to choose reply text that conveys specific system-dependent information, when appropriate.

RFC-959 File Transfer Protocol (FTP) -- Reply Codes

Reply Codes by Function Groups

200 Command okay.
500 Syntax error, command unrecognized. This may include errors such as command line too long.
501 Syntax error in parameters or arguments.
202 Command not implemented, superfluous at this site.
502 Command not implemented.
503 Bad sequence of commands.
504 Command not implemented for that parameter.

110 Restart marker reply.
    110 MARK ssss = rrrr
Here:
* ssss is a text string that appeared in a Restart Marker in the data stream and encodes a position in the sender's file system;
* rrrr encodes the corresponding position in the receiver's file system.

The encoding, which is specific to a particular file system and network implementation, is always generated and interpreted by the same system, either sender or receiver.
211 System status, or system help reply.
212 Directory status.
213 File status.
214 Help message.
On how to use the server or the meaning of a particular non-standard command. This reply is useful only to the human user.
215 NAME system type.
Where NAME is an official system name from the list in the Assigned Numbers document.

120 Service ready in nnn minutes.
220 Service ready for new user.
221 Service closing control connection.
Logged out if appropriate.
421 Service not available, closing control connection.
This may be a reply to any command if the service knows it must shut down.
125 Data connection already open; transfer starting.
225 Data connection open; no transfer in progress.
425 Can't open data connection.
226 Closing data connection.
Requested file action successful (for example, file transfer or file abort).
426 Connection closed; transfer aborted.
227 Entering Passive Mode (h1,h2,h3,h4,p1,p2).
230 User logged in, proceed.
530 Not logged in.
331 User name okay, need password.
332 Need account for login.
532 Need account for storing files.

150 File status okay; about to open data connection.
250 Requested file action okay, completed.
257 "PATHNAME" created.

350 Requested file action pending further information.
450 Requested file action not taken.
File unavailable (e.g., file busy).
550 Requested action not taken.
File unavailable (e.g., file not found, no access).
451 Requested action aborted. Local error in processing.
551 Requested action aborted. Page type unknown.
452 Requested action not taken.
Insufficient storage space in system.
552 Requested file action aborted.
Exceeded storage allocation (for current directory or dataset).
553 Requested action not taken.
File name not allowed.

RFC-959 File Transfer Protocol (FTP) -- Reply Codes

Numeric Order List of Reply Codes

110 Restart marker reply.
In this case, the text is exact and not left to the particular implementation; it must read:
MARK yyyy = mmmm
Where yyyy is User-process data stream marker, and mmmm server's equivalent marker (note the spaces between markers and "=").
120 Service ready in nnn minutes.
125 Data connection already open; transfer starting.
150 File status okay; about to open data connection.
200 Command okay.
202 Command not implemented, superfluous at this site.
211 System status, or system help reply.
212 Directory status.
213 File status.
214 Help message.
On how to use the server or the meaning of a particular non-standard command.
This reply is useful only to the human user.
215 NAME system type.
Where NAME is an official system name from the list in the Assigned Numbers document.
220 Service ready for new user.
221 Service closing control connection.
Logged out if appropriate.
225 Data connection open; no transfer in progress.
226 Closing data connection.
Requested file action successful (for example, file transfer or file abort).
227 Entering Passive Mode (h1,h2,h3,h4,p1,p2).
230 User logged in, proceed.
250 Requested file action okay, completed.
257 "PATHNAME" created.
331 User name okay, need password.
332 Need account for login.
350 Requested file action pending further information.
421 Service not available, closing control connection.
This may be a reply to any command if the service knows it must shut down.
425 Can't open data connection.
426 Connection closed; transfer aborted.
450 Requested file action not taken.
File unavailable (e.g., file busy).
451 Requested action aborted: local error in processing.
452 Requested action not taken.
Insufficient storage space in system.
500 Syntax error, command unrecognized.
This may include errors such as command line too long.
501 Syntax error in parameters or arguments.
502 Command not implemented.
503 Bad sequence of commands.
504 Command not implemented for that parameter.
530 Not logged in.

532 Need account for storing files.
550 Requested action not taken.
File unavailable (e.g., file not found, no access).
551 Requested action aborted: page type unknown.
552 Requested file action aborted.
Exceeded storage allocation (for current directory or dataset).
553 Requested action not taken.
File name not allowed.

RFC-959 File Transfer Protocol (FTP)

Declarative Specifications

Minimum Implementation

The following commands and options **must** be supported by every server-FTP and user-FTP, except in cases where the underlying file system or operating system does not allow or support a particular command. This list reflects the requirements as of January 1991.

Type:	ASCII Non-print, IMAGE, LOCAL 8
Mode:	Stream
Structure:	File, Record
Commands:	USER, PASS, ACCT, PORT, PASV, TYPE, MODE, STRU, RETR, STOR, APPE, RNFR, RNTO, DELE, CWD, CDUP, RMD, MKD, PWD, LIST, NLST, SYST, STAT, HELP, NOOP, QUIT.

Discussion

Vendors are encouraged to implement a larger subset of the protocol. For example, there are important robustness features in the protocol (e.g., Restart, ABOR, block mode) that would be an aid to some Internet users but are not widely implemented.

A host that does not have record structures in its file system may still accept files with STRU R, recording the byte stream literally.

The default values for transfer parameters are:

TYPE - ASCII Non-print
MODE - Stream
STRU - File

All hosts must accept the above as the standard defaults.

RFC-959 File Transfer Protocol (FTP) -- Declarative Specifications

Connections

The server protocol interpreter shall "listen" on Port L. The user or user protocol interpreter shall initiate the full-duplex control connection. Server- and user- processes should follow the conventions of the Telnet protocol as in RFC-854 "Telnet Protocol", with the exception that Telnet Option Negotiation is not used. Servers are under no obligation to provide for editing of command lines and may require that it be done in the user host. The control connection shall be closed by the server at the user's request after all transfers and replies are completed.

On a multihomed server host, the default data transfer port (L-1) **must** be associated with the same local IP address as the corresponding control connection to port L.

A user-FTP **must not** send any Telnet controls other than SYNCH and IP on an FTP control connection. In particular, it **must not** attempt to negotiate Telnet options on the control connection. However, a server-FTP **must** be capable of accepting and refusing Telnet negotiations (i.e., sending DONT/WONT).

The user-DTP must "listen" on the specified data port; this may be the default user port (U) or a port specified in the PORT command. The server shall initiate the data connection from his own default data port (L-1) using the specified user data port. The direction of the transfer will be determined by the FTP service command.

Note that all FTP implementation must support data transfer using the default port, and that only the USER-PI may initiate the use of non-default ports.

When data is to be transferred between two servers, A and B (refer to Figure 2), the user-PI, C, sets up control connections with both server-PI's. One of the servers, say A, is then sent a PASV command telling him to "listen" on his data port rather than initiate a connection when he receives a transfer service command. When the user-PI receives an acknowledgment to the PASV command, which includes the identity of the host and port being listened on, the user-PI then sends A's port, a, to B in a PORT command; a reply is returned. The user-PI may then send the corresponding service commands to A and B. Server B initiates the connection and the transfer proceeds. The command-reply sequence is listed below where the messages are vertically synchronous but horizontally asynchronous:

User-PI - Server A	**User-PI - Server B**
C->A : Connect	C->B : Connect
C->A : PASV	
A->C : 227 Entering Passive Mode. A1,A2,A3,A4,a1,a2	
	C->B : PORT A1,A2,A3,A4,a1,a2
	B->C : 200 Okay
C->A : STOR	C->B : RETR
B->A : Connect to HOST-A, PORT-a	

Figure 3

The data connection shall be closed by the server under the conditions described in the Section on Establishing Data Connections. If the data connection is to be closed following a data transfer where closing the connection is not required to indicate the end-of-file, the server must do so immediately. Waiting until after a new transfer command is not permitted because the user-process will have already tested the data connection to see if it needs to do a "listen"; (remember that the user must "listen" on a closed data port BEFORE sending the transfer request). To prevent a race condition here, the server sends a reply (226) after

closing the data connection (or if the connection is left open, a "file transfer completed" reply (250) and the user-PI should wait for one of these replies before issuing a new transfer command).

Any time either the user or server see that the connection is being closed by the other side, it should promptly read any remaining data queued on the connection and issue the close on its own side.

RFC-959 File Transfer Protocol (FTP) -- Declarative Specifications

Commands

The commands are Telnet character strings transmitted over the control connections as described in FTP Commands. The command functions and semantics are described in Access Control Commands, Transfer Parameter Commands, and FTP Service Commands. The command syntax is specified here.

The commands begin with a command code followed by an argument field. The command codes are four or fewer alphabetic characters. Upper and lower case alphabetic characters are to be treated identically. Thus, any of the following may represent the retrieve command:

RETR Retr retr ReTr rETr

This also applies to any symbols representing parameter values, such as A or a for ASCII TYPE. The command codes and the argument fields are separated by one or more spaces.

The argument field consists of a variable length character string ending with the character sequence <CRLF> (Carriage Return, Line Feed) for NVT-ASCII representation; for other negotiated languages a different end of line character might be used. It should be noted that the server is to take no action until the end of line code is received.

The syntax is specified below in NVT-ASCII. All characters in the argument field are ASCII characters including any ASCII represented decimal integers. Square brackets denote an optional argument field. If the option is not taken, the appropriate default is implied.

FTP Commands

The following are the FTP commands:

```
USER <SP> <username> <CRLF>
PASS <SP> <password> <CRLF>
ACCT <SP> <account-information> <CRLF>
CWD  <SP> <pathname> <CRLF>
CDUP <CRLF>
SMNT <SP> <pathname> <CRLF>
QUIT <CRLF>
REIN <CRLF>
PORT <SP> <host-port> <CRLF>
PASV <CRLF>
TYPE <SP> <type-code> <CRLF>
STRU <SP> <structure-code> <CRLF>
MODE <SP> <mode-code> <CRLF>
RETR <SP> <pathname> <CRLF>
STOR <SP> <pathname> <CRLF>
STOU <CRLF>
APPE <SP> <pathname> <CRLF>
ALLO <SP> <decimal-integer>
    [<SP> R <SP> <decimal-integer>] <CRLF>
REST <SP> <marker> <CRLF>
RNFR <SP> <pathname> <CRLF>
RNTO <SP> <pathname> <CRLF>
ABOR <CRLF>
DELE <SP> <pathname> <CRLF>
RMD  <SP> <pathname> <CRLF>
MKD  <SP> <pathname> <CRLF>
PWD  <CRLF>
LIST [<SP> <pathname>] <CRLF>
NLST [<SP> <pathname>] <CRLF>
SITE <SP> <string> <CRLF>
SYST <CRLF>
STAT [<SP> <pathname>] <CRLF>
HELP [<SP> <string>] <CRLF>
NOOP <CRLF>
```

FTP Command Arguments

The syntax of the above argument fields (using BNF notation where applicable) is:

```
<username> ::= <string>
<password> ::= <string>
<account-information> ::= <string>
<string> ::= <char> | <char><string>
<char> ::= any of the 128 ASCII characters except <CR> and
<LF>
<marker> ::= <pr-string>
<pr-string> ::= <pr-char> | <pr-char><pr-string>
<pr-char> ::= printable characters, any
                ASCII code 33 through 126
<byte-size> ::= <number>
<host-port> ::= <host-number>,<port-number>
<host-number> ::= <number>,<number>,<number>,<number>
<port-number> ::= <number>,<number>
<number> ::= any decimal integer 1 through 255
<form-code> ::= N | T | C
<type-code> ::= A [<sp> <form-code>]
                | E [<sp> <form-code>]
                | I
                | L <sp> <byte-size>
<structure-code> ::= F | R | P
<mode-code> ::= S | B | C
<pathname> ::= <string>
<decimal-integer> ::= any decimal integer
```

RFC-959 File Transfer Protocol (FTP) -- Declarative Specifications

Sequencing Of Commands And Replies

The communication between the user and server is intended to be an alternating dialogue. As such, the user issues an FTP command and the server responds with a prompt primary reply. The user should wait for this initial primary success or failure response before sending further commands.

Certain commands require a second reply for which the user should also wait. These replies may, for example, report on the progress or completion of file transfer or the closing of the data connection. They are secondary replies to file transfer commands.

One important group of informational replies is the connection greetings. Under normal circumstances, a server will send a 220 reply, "awaiting input", when the connection is completed. The user should wait for this greeting message before sending any commands. If the server is unable to accept input right away, a 120 "expected delay" reply should be sent immediately and a 220 reply when ready. The user will then know not to hang up if there is a delay.

Spontaneous Replies

Sometimes "the system" spontaneously has a message to be sent to a user (usually all users). For example, "System going down in 15 minutes". There is no provision in FTP for such spontaneous information to be sent from the server to the user. It is recommended that such information be queued in the server-PI and delivered to the user-PI in the next reply (possibly making it a multi-line reply).

The table below lists alternative success and failure replies for each command. These must be strictly adhered to; a server may substitute text in the replies, but the meaning and action implied by the code numbers and by the specific command reply sequence cannot be altered.

Command-Reply Sequences

In this section, the command-reply sequence is presented. Each command is listed with its possible replies; command groups are listed together. Preliminary replies are listed first (with their succeeding replies indented and under them), then positive and negative completion, and finally intermediary replies with the remaining commands from the sequence following. This listing forms the basis for the state diagrams, which will be presented separately.

Connection Establishment

- 120
  - 220
- 220
- 421

Login

- USER
  - 230
  - 530
  - 500, 501, 421
  - 331, 332
- PASS
  - 230
  - 202

530
500, 501, 503, 421
332

ACCT
230
202
530
500, 501, 503, 421

CWD
250
500, 501, 502, 421, 530, 550

CDUP
200
500, 501, 502, 421, 530, 550

SMNT
202, 250
500, 501, 502, 421, 530, 550

Logout

REIN
120
220
220
421
500, 502

QUIT
221
500

Transfer parameters

PORT
200
500, 501, 421, 530

PASV
227
500, 501, 502, 421, 530

MODE
200
500, 501, 504, 421, 530

TYPE
200
500, 501, 504, 421, 530

STRU
200
500, 501, 504, 421, 530

File action commands

ALLO
200
202
500, 501, 504, 421, 530

REST
500, 501, 502, 421, 530
350

STOR
125, 150

(110)
226, 250
425, 426, 451, 551, 552
532, 450, 452, 553
500, 501, 421, 530
STOU
125, 150
(110)
226, 250
425, 426, 451, 551, 552
532, 450, 452, 553
500, 501, 421, 530
RETR
125, 150
(110)
226, 250
425, 426, 451
450, 550
500, 501, 421, 530
LIST
125, 150
226, 250
425, 426, 451
450
500, 501, 502, 421, 530
NLST
125, 150
226, 250
425, 426, 451
450
500, 501, 502, 421, 530
APPE
125, 150
(110)
226, 250
425, 426, 451, 551, 552
532, 450, 550, 452, 553
500, 501, 502, 421, 530
RNFR
450, 550
500, 501, 502, 421, 530
350
RNTO
250
532, 553
500, 501, 502, 503, 421, 530
DELE
250
450, 550
500, 501, 502, 421, 530
RMD
250
500, 501, 502, 421, 530, 550
MKD
257

500, 501, 502, 421, 530, 550

PWD

257

500, 501, 502, 421, 550

ABOR

225, 226

500, 501, 502, 421

Informational commands

SYST

215

500, 501, 502, 421

STAT

211, 212, 213

450

500, 501, 502, 421, 530

HELP

211, 214

500, 501, 502, 421

Miscellaneous commands

SITE

200

202

500, 501, 530

NOOP

200

500 421

RFC-959 File Transfer Protocol (FTP)

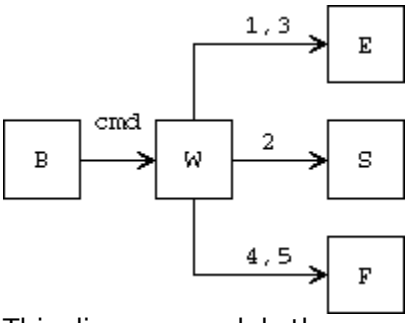
State Diagrams

Here we present state diagrams for a very simple minded FTP implementation. Only the first digit of the reply codes is used. There is one state diagram for each group of FTP commands or command sequences.

The command groupings were determined by constructing a model for each command then collecting together the commands with structurally identical models.

For each command or command sequence there are three possible outcomes: success (S), failure (F), and error (E). In the state diagrams below we use the symbol B for "begin", and the symbol W for "wait for reply".

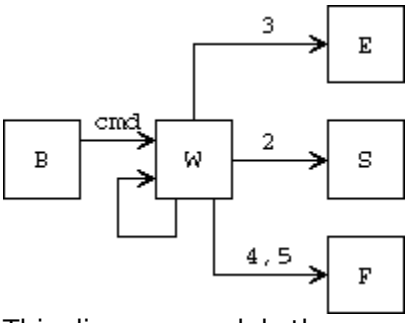
We first present the diagram that represents the largest group of FTP commands:



This diagram models the commands:

ABOR, ALLO, DELE, CWD, CDUP, SMNT, HELP, MODE, NOOP, PASV, QUIT, SITE, PORT, SYST, STAT, RMD, MKD, PWD, STRU, and TYPE.

The other large group of commands is represented by a very similar diagram:

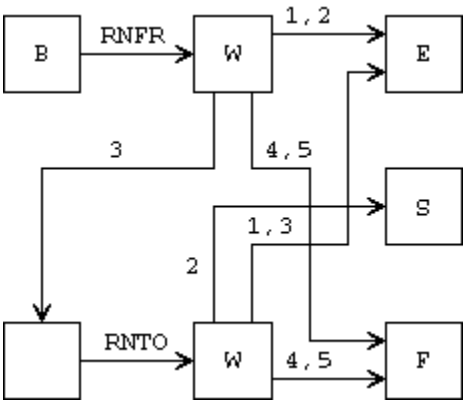


This diagram models the commands:

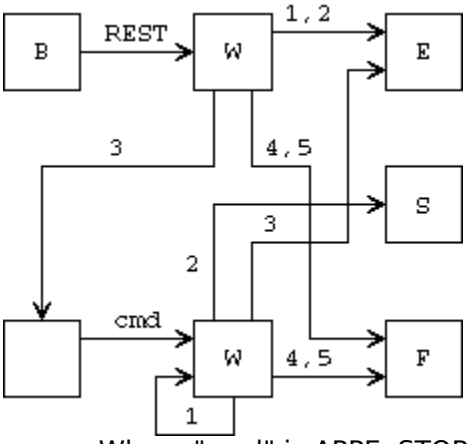
APPE, LIST, NLST, REIN, RETR, STOR, and STOU.

Note that this second model could also be used to represent the first group of commands, the only difference being that in the first group the 100 series replies are unexpected and therefore treated as error, while the second group expects (some may require) 100 series replies. Remember that at most, one 100 series reply is allowed per command.

The remaining diagrams model command sequences, perhaps the simplest of these is the rename sequence:



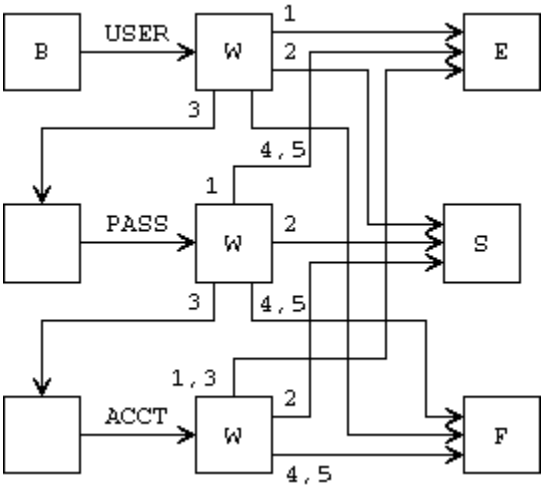
The next diagram is a simple model of the Restart command:



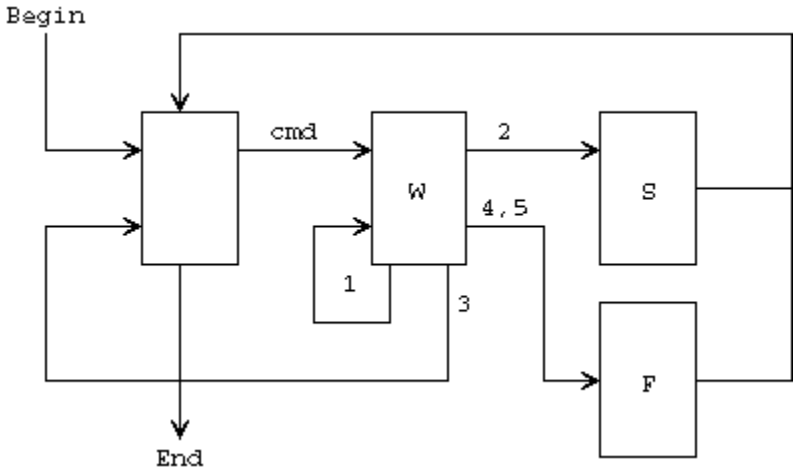
Where "cmd" is APPE, STOR, or RETR.

We note that the above three models are similar. The Restart differs from the Rename two only in the treatment of 100 series replies at the second stage, while the second group expects (some may require) 100 series replies. Remember that at most, one 100 series reply is allowed per command.

The most complicated diagram is for the Login sequence:



Finally, we present a generalized diagram that could be used to model the command and reply interchange:


Begin
cmd
W
2
S
4, 5
1
3
F
End

RFC-959 File Transfer Protocol (FTP)

Typical FTP Scenario

User at host U wanting to transfer files to/from host S: In general, the user will communicate to the server via a mediating user-FTP process. The following may be a typical scenario. The user-FTP prompts are shown in parentheses, '---->' represents commands from host U to host S, and '<----' represents replies from host S to host U.

Local Commands By User	**Action Involved**
ftp (host) multics<CR>	Connect to host S, port L, establishing control connections. <---- 220 Service ready <CRLF>.
username Doe <CR>	USER Doe<CRLF>----> <---- 331 User name ok, need password<CRLF>.
password mumble <CR>	PASS mumble<CRLF>----> <---- 230 User logged in<CRLF>.
retrieve (local type) ASCII<CR> (local pathname) test 1 <CR> (for. pathname) test.pl1<CR>	 User-FTP opens local file in ASCII. RETR test.pl1<CRLF> ---->
<----150 File status okay; about to open data connection<CRLF>. Server makes data connection to port U. <---- 226 Closing data connection, file transfer successful<CRLF>.	
type Image<CR>	TYPE I<CRLF> ----> <---- 200 Command OK<CRLF>
store (local type) image<CR> (local pathname) file dump<CR> (for.pathname) >ud>cn>fd<CR>	 User-FTP opens local file in Image. STOR >udd>cn>fd<CRLF> ----> <---- 550 Access denied<CRLF>
terminate	QUIT <CRLF> ----> Server closes all connections.

Connection Establishment

The FTP control connection is established via TCP between the user process port U and the server process port L. This protocol is assigned the service port 21 (25 octal), that is L=21.

Appendix I - Page Structure

The need for FTP to support page structure derives principally from the need to support efficient transmission of files between TOPS-20 systems, particularly the files used by NLS.

Implementation of page structure is **not recommended** in general. However, if a host system does need to implement FTP for "random access" or "holey" files, it **must** use the defined page structure format rather than define a new private FTP format.

The file system of TOPS-20 is based on the concept of pages. The operating system is most efficient at manipulating files as pages. The operating system provides an interface to the file system so that many applications view files as sequential streams of characters. However, a few applications use the underlying page structures directly, and some of these create holey files.

A TOPS-20 disk file consists of four things: a pathname, a page table, a (possibly empty) set of pages, and a set of attributes.

The pathname is specified in the RETR or STOR command. It includes the directory name, file name, file name extension, and generation number.

The page table contains up to 2**18 entries. Each entry may be EMPTY, or may point to a page. If it is not empty, there are also some page-specific access bits; not all pages of a file need have the same access protection.

> A page is a contiguous set of 512 words of 36 bits each.

The attributes of the file, in the File Descriptor Block (FDB), contain such things as creation time, write time, read time, writer's byte-size, end-of-file pointer, count of reads and writes, backup system tape numbers, etc.

Note that there is NO requirement that entries in the page table be contiguous. There may be empty page table slots between occupied ones. Also, the end of file pointer is simply a number. There is no requirement that it in fact point at the "last" datum in the file. Ordinary sequential I/O calls in TOPS-20 will cause the end of file pointer to be left after the last datum written, but other operations may cause it not to be so, if a particular programming system so requires.

In fact, in both of these special cases, "holey" files and end-of-file pointers NOT at the end of the file, occur with NLS data files.

The TOPS-20 paged files can be sent with the FTP transfer parameters: TYPE L 36, STRU P, and MODE S (in fact, any mode could be used).

Each page of information has a header. Each header field, which is a logical byte, is a TOPS-20 word, since the TYPE is L 36.

The header fields are:

Word 0: Header Length.

> The header length is 5.

Word 1: Page Index.

> If the data is a disk file page, this is the number of that page in the file's page map. Empty pages (holes) in the file are simply not sent. Note that a hole is NOT the same as a page of zeros.

Word 2: Data Length.

The number of data words in this page, following the header. Thus, the total length of the transmission unit is the Header Length plus the Data Length.

Word 3: Page Type.

A code for what type of chunk this is. A data page is type 3, the FDB page is type 2.

Word 4: Page Access Control.

The access bits associated with the page in the file's page map. (This full word quantity is put into AC2 of an SPACS by the program reading from net to disk.)

After the header are Data Length data words. Data Length is currently either 512 for a data page or 31 for an FDB. Trailing zeros in a disk file page may be discarded, making Data Length less than 512 in that case.

Appendix II - Directory Commands

Since UNIX has a tree-like directory structure in which directories are as easy to manipulate as ordinary files, it is useful to expand the FTP servers on these machines to include commands which deal with the creation of directories. Since there are other hosts on the ARPA-Internet which have tree-like directories (including TOPS-20 and Multics), these commands are as general as possible.

Four directory commands have been added to FTP:

MKD pathname

Make a directory with the name "pathname".

RMD pathname

Remove the directory with the name "pathname".

PWD

Print the current working directory name.

CDUP

Change to the parent of the current working directory.

The "pathname" argument should be created (removed) as a subdirectory of the current working directory, unless the "pathname" string contains sufficient information to specify otherwise to the server, e.g., "pathname" is an absolute pathname (in UNIX and Multics), or pathname is something like "<abso.lute.path>" to TOPS-20.

Reply Codes

The CDUP command is a special case of CWD, and is included to simplify the implementation of programs for transferring directory trees between operating systems having different syntaxes for naming the parent directory. The reply codes for CDUP be identical to the reply codes of CWD.

The reply codes for RMD be identical to the reply codes for its file analogue, DELE.

The reply codes for MKD, however, are a bit more complicated. A freshly created directory will probably be the object of a future CWD command. Unfortunately, the argument to MKD may not always be a suitable argument for CWD. This is the case, for example, when a TOPS-20 subdirectory is created by giving just the subdirectory name. That is, with a TOPS-20 server FTP, the command sequence

MKD MYDIR
CWD MYDIR

will fail. The new directory may only be referred to by its "absolute" name; e.g., if the MKD command above were issued while connected to the directory <DFRANKLIN>, the new subdirectory could only be referred to by the name <DFRANKLIN.MYDIR>.

Even on UNIX and Multics, however, the argument given to MKD may not be suitable. If it is a "relative" pathname (i.e., a pathname which is interpreted relative to the current directory), the user would need to be in the same current directory in order to reach the subdirectory. Depending on the application, this may be inconvenient. It is not very robust in any case.

To solve these problems, upon successful completion of an MKD command, the server should

return a line of the form:

257<space>"<directory-name>"<space><commentary>

That is, the server will tell the user what string to use when referring to the created directory. The directory name can contain any character; embedded double-quotes should be escaped by double-quotes (the "quote-doubling" convention).

For example, a user connects to the directory /usr/dm, and creates a subdirectory, named pathname:

CWD /usr/dm
200 directory changed to /usr/dm

MKD pathname
257 "/usr/dm/pathname" directory created

An example with an embedded double quote:

MKD foo"bar
257 "/usr/dm/foo""bar" directory created

CWD /usr/dm/foo"bar
200 directory changed to /usr/dm/foo"bar

The prior existence of a subdirectory with the same name is an error, and the server must return an "access denied" error reply in that case.

CWD /usr/dm
200 directory changed to /usr/dm

MKD pathname
521-"/usr/dm/pathname" directory already exists;
521 taking no action.

The failure replies for MKD are analogous to its file creating cousin, STOR. Also, an "access denied" return is given if a file name with the same name as the subdirectory will conflict with the creation of the subdirectory (this is a problem on UNIX, but shouldn't be one on TOPS-20).

Essentially because the PWD command returns the same type of information as the successful MKD command, the successful PWD command uses the 257 reply code as well.

Subtleties

Because these commands will be most useful in transferring subtrees from one machine to another, carefully observe that the argument to MKD is to be interpreted as a sub-directory of the current working directory, unless it contains enough information for the destination host to tell otherwise. A hypothetical example of its use in the TOPS-20 world:

CWD <some.where>
200 Working directory changed

MKD overrainbow
257 "<some.where.overrainbow>" directory created

CWD overrainbow
431 No such directory

CWD <some.where.overrainbow>
200 Working directory changed

CWD <some.where>
200 Working directory changed to <some.where>

MKD <unambiguous>

257 "<unambiguous>" directory created

CWD <unambiguous>

Note that the first example results in a subdirectory of the connected directory. In contrast, the argument in the second example contains enough information for TOPS-20 to tell that the <unambiguous> directory is a top-level directory. Note also that in the first example the user "violated" the protocol by attempting to access the freshly created directory with a name other than the one returned by TOPS-20. Problems could have resulted in this case had there been an <overrainbow> directory; this is an ambiguity inherent in some TOPS-20 implementations. Similar considerations apply to the RMD command. The point is this: except where to do so would violate a host's conventions for denoting relative versus absolute pathnames, the host should treat the operands of the MKD and RMD commands as subdirectories. The 257 reply to the MKD command must always contain the absolute pathname of the created directory.

Appendix III - RFCs On FTP

Bhushan, Abhay, "A File Transfer Protocol", RFC 114 (NIC 5823), MIT-Project MAC, 16 April 1971.

Harslem, Eric, and John Heafner, "Comments on RFC 114 (A File Transfer Protocol)", RFC 141 (NIC 6726), RAND, 29 April 1971.

Bhushan, Abhay, et al, "The File Transfer Protocol", RFC 172 (NIC 6794), MIT-Project MAC, 23 June 1971.

Braden, Bob, "Comments on DTP and FTP Proposals", RFC 238 (NIC 7663), UCLA/CCN, 29 September 1971.

Bhushan, Abhay, et al, "The File Transfer Protocol", RFC 265 (NIC 7813), MIT-Project MAC, 17 November 1971.

McKenzie, Alex, "A Suggested Addition to File Transfer Protocol", RFC 281 (NIC 8163), BBN, 8 December 1971.

Bhushan, Abhay, "The Use of "Set Data Type" Transaction in File Transfer Protocol", RFC 294 (NIC 8304), MIT-Project MAC, 25 January 1972.

Bhushan, Abhay, "The File Transfer Protocol", RFC 354 (NIC 10596), MIT-Project MAC, 8 July 1972.

Bhushan, Abhay, "Comments on the File Transfer Protocol (RFC 354)", RFC 385 (NIC 11357), MIT-Project MAC, 18 August 1972.

Hicks, Greg, "User FTP Documentation", RFC 412 (NIC 12404), Utah, 27 November 1972.

Bhushan, Abhay, "File Transfer Protocol (FTP) Status and Further Comments", RFC 414 (NIC 12406), MIT-Project MAC, 20 November 1972.

Braden, Bob, "Comments on File Transfer Protocol", RFC 430 (NIC 13299), UCLA/CCN, 7 February 1973.

Thomas, Bob, and Bob Clements, "FTP Server-Server Interaction", RFC 438 (NIC 13770), BBN, 15 January 1973.

Braden, Bob, "Print Files in FTP", RFC 448 (NIC 13299), UCLA/CCN, 27 February 1973.

McKenzie, Alex, "File Transfer Protocol", RFC 454 (NIC 14333), BBN, 16 February 1973.

Bressler, Bob, and Bob Thomas, "Mail Retrieval via FTP", RFC 458 (NIC 14378), BBN-NET and BBN-TENEX, 20 February 1973.

Neigus, Nancy, "File Transfer Protocol", RFC 542 (NIC 17759), BBN, 12 July 1973.

Krilanovich, Mark, and George Gregg, "Comments on the File Transfer Protocol", RFC 607 (NIC 21255), UCSB, 7 January 1974.

Pogran, Ken, and Nancy Neigus, "Response to RFC 607 - Comments on the File Transfer Protocol", RFC 614 (NIC 21530), BBN, 28 January 1974.

Krilanovich, Mark, George Gregg, Wayne Hathaway, and Jim White, "Comments on the File Transfer Protocol", RFC 624 (NIC 22054), UCSB, Ames Research Center, SRI-ARC, 28 February 1974.

Bhushan, Abhay, "FTP Comments and Response to RFC 430", RFC 463 (NIC 14573), MIT-DMCG, 21 February 1973.

Braden, Bob, "FTP Data Compression", RFC 468 (NIC 14742), UCLA/CCN, 8 March 1973.

Bhushan, Abhay, "FTP and Network Mail System", RFC 475 (NIC 14919), MIT-DMCG, 6 March

1973.

Bressler, Bob, and Bob Thomas "FTP Server-Server Interaction - II", RFC 478 (NIC 14947), BBN-NET and BBN-TENEX, 26 March 1973.

White, Jim, "Use of FTP by the NIC Journal", RFC 479 (NIC 14948), SRI-ARC, 8 March 1973.

White, Jim, "Host-Dependent FTP Parameters", RFC 480 (NIC 14949), SRI-ARC, 8 March 1973.

Padlipsky, Mike, "An FTP Command-Naming Problem", RFC 506 (NIC 16157), MIT-Multics, 26 June 1973.

Day, John, "Memo to FTP Group (Proposal for File Access Protocol)", RFC 520 (NIC 16819), Illinois, 25 June 1973.

Merryman, Robert, "The UCSD-CC Server-FTP Facility", RFC 532 (NIC 17451), UCSD-CC, 22 June 1973.

Braden, Bob, "TENEX FTP Problem", RFC 571 (NIC 18974), UCLA/CCN, 15 November 1973.

McKenzie, Alex, and Jon Postel, "Telnet and FTP Implementation - Schedule Change", RFC 593 (NIC 20615), BBN and MITRE, 29 November 1973.

Sussman, Julie, "FTP Error Code Usage for More Reliable Mail Service", RFC 630 (NIC 30237), BBN, 10 April 1974.

Postel, Jon, "Revised FTP Reply Codes", RFC 640 (NIC 30843), UCLA/NMC, 5 June 1974.

Harvey, Brian, "Leaving Well Enough Alone", RFC 686 (NIC 32481), SU-AI, 10 May 1975.

Harvey, Brian, "One More Try on the FTP", RFC 691 (NIC 32700), SU-AI, 28 May 1975.

Lieb, J., "CWD Command of FTP", RFC 697 (NIC 32963), 14 July 1975.

Harrenstien, Ken, "FTP Extension: XSEN", RFC 737 (NIC 42217), SRI-KL, 31 October 1977.

Harrenstien, Ken, "FTP Extension: XRSQ/XRCP", RFC 743 (NIC 42758), SRI-KL, 30 December 1977.

Lebling, P. David, "Survey of FTP Mail and MLFL", RFC 751, MIT, 10 December 1978.

Postel, Jon, "File Transfer Protocol Specification", RFC 765, ISI, June 1980.

Mankins, David, Dan Franklin, and Buzz Owen, "Directory Oriented FTP Commands", RFC 776, BBN, December 1980.

Padlipsky, Michael, "FTP Unique-Named Store Command", RFC 949, MITRE, July 1985.

REFERENCES

[1] Feinler, Elizabeth, "Internet Protocol Transition Workbook", Network Information Center, SRI International, March 1982.

[2] Postel, Jon, "Transmission Control Protocol - DARPA Internet Program Protocol Specification", RFC 793, DARPA, September 1981.

[3] Postel, Jon, and Joyce Reynolds, "Telnet Protocol Specification", RFC 854, ISI, May 1983.

[4] Reynolds, Joyce, and Jon Postel, "Assigned Numbers", RFC 943, ISI, April 1985.

RFC-974--Mail Routing and the Domain System

Craig Partridge
CSNET CIC BBN Laboratories Inc
January 1986

This RFC presents a description of how mail systems on the Internet are expected to route messages based on information from the domain system described in RFCs 1034 and 1035.

RFC-974--Mail Routing and the Domain System

Introduction

The purpose of this memo is to explain how mailers are to decide how to route a message addressed to a given Internet domain name. This involves a discussion of how mailers interpret MX RRs, which are used for message routing. Note that this memo makes no statement about how mailers are to deal with MB and MG RRs, which are used for interpreting mailbox names.

Under the Domain System, certain assumptions about mail addresses have been changed. Previously, one could usually assume that if a message was addressed to a mailbox, for example, at LOKI.BBN.COM, that one could just open an SMTP connection to LOKI.BBN.COM and pass the message along. This system broke down in certain situations, such as for certain UUCP and CSNET hosts which were not directly attached to the Internet, but these hosts could be handled as special cases in configuration files (for example, most mailers were set up to automatically forward mail addressed to a CSNET host to CSNET-RELAY.ARPA).

Under domains, one cannot simply open a connection to LOKI.BBN.COM, but must instead ask the domain system where messages to LOKI.BBN.COM are to be delivered. And the domain system may direct a mailer to deliver messages to an entirely different host, such as SH.CS.NET. Or, in a more complicated case, the mailer may learn that it has a choice of routes to LOKI.BBN.COM. This memo is essentially a set of guidelines on how mailers should behave in this more complex world.

Readers are expected to be familiar with RFC-1034 and RFC-1035.

RFC-974--Mail Routing and the Domain System

What the Domain Servers Know

The domain servers store information as a series of resource records (RRs), each of which contains a particular piece of information about a given domain name (which is usually, but not always, a host). The simplest way to think of a RR is as a typed pair of datum, a domain name matched with relevant data, and stored with some additional type information to help systems determine when the RR is relevant. For the purposes of message routing, the system stores RRs known as MX RRs. Each MX matches a domain name with two pieces of data, a preference value (an unsigned 16-bit integer), and the name of a host. The preference number is used to indicate in what order the mailer should attempt deliver to the MX hosts, with the lowest numbered MX being the one to try first. Multiple MXs with the same preference are permitted and have the same priority.

In addition to mail information, the servers store certain other types of RR's which mailers may encounter or choose to use. These are: the canonical name (CNAME) RR, which simply states that the domain name queried for is actually an alias for another domain name,which is the proper, or canonical, name; and the Well Known Service (WKS) RR, which stores information about network services (such as SMTP) a given domain name supports.

RFC-974--Mail Routing and the Domain System

General Routing Guidelines

Before delving into a detailed discussion of how mailers are expected to do mail routing, it would seem to make sense to give a brief overview of how this memo is approaching the problems that routing poses.

The first major principle is derived from the definition of the preference field in MX records, and is intended to prevent mail looping. If the mailer is on a host which is listed as an MX for the destination host, the mailer may only deliver to an MX which has a lower preference count than its own host.

It is also possible to cause mail looping because routing information is out of date or incomplete. Out of date information is only a problem when domain tables are changed. The changes will not be known to all affected hosts until their resolver caches time out. There is no way to ensure that this will not happen short of requiring mailers and their resolvers to always send their queries to an authoritative server, and never use data stored in a cache. This is an impractical solution, since eliminating resolver caching would make mailing inordinately expensive. What is more, the out-of-date RR problem should not happen if, when a domain table is changed, affected hosts (those in the list of MXs) have their resolver caches flushed. In other words, given proper precautions, mail looping as a result of domain information should be avoidable, without requiring mailers to query authoritative servers. (The appropriate precaution is to check with a host's administrator before adding that host to a list of MXs).

The incomplete data problem also requires some care when handling domain queries. If the answer section of a query is incomplete critical MX RRs may be left out. This may result in mail looping, or in a message being mistakenly labelled undeliverable. As a result, mailers may only accept responses from the domain system which have complete answer sections. Note that this entire problem can be avoided by only using virtual circuits for queries, but since this situation is likely to be very rare and datagrams are the preferred way to interact with the domain system, implementors should probably just ensure that their mailer will repeat a query with virtual circuits should the truncation bit ever be set.

Determining Where to Send a Message

The explanation of how mailers should decide how to route a message is discussed in terms of the problem of a mailer on a host with domain name LOCAL trying to deliver a message addressed to the domain name REMOTE. Both LOCAL and REMOTE are assumed to be syntactically correct domain names. Furthermore, LOCAL is assumed to be the official name for the host on which the mailer resides (i.e., it is not a alias).

Issuing a Query

The first step for the mailer at LOCAL is to issue a query for MX RRs for REMOTE. It is strongly urged that this step be taken every time a mailer attempts to send the message. The hope is that changes in the domain database will rapidly be used by mailers, and thus domain administrators will be able to re-route in-transit messages for defective hosts by simply changing their domain databases.

Certain responses to the query are considered errors:

- Getting no response to the query. The domain server the mailer queried never sends anything back. (This is distinct from an answer which contains no answers to the query, which is not an error).

- Getting a response in which the truncation field of the header is Mail Routing and the Domain System set. (Recall discussion of incomplete queries above). Mailers may not use responses of this type, and should repeat the query using virtual circuits instead of datagrams.

- Getting a response in which the response code is non-zero.

Mailers are expected to do something reasonable in the face of an error. The behaviour for each type of error is not specified here, but implementors should note that different types of errors should probably be treated differently. For example, a response code of "non-existent domain" should probably cause the message to be returned to the sender as invalid, while a response code of "server failure" should probably cause the message to be retried later.

There is one other special case. If the response contains an answer which is a CNAME RR, it indicates that REMOTE is actually an alias for some other domain name. The query should be repeated with the canonical domain name.

If the response does not contain an error response, and does not contain aliases, its answer section should be a (possibly zero length) list of MX RRs for domain name REMOTE (or REMOTE's true domain name if REMOTE was a alias). The next section describes how this list is interpreted.

Interpreting the List of MX RRs

NOTE: This section only discusses how mailers choose which names to try to deliver a message to, working from a list of RR's. It does not discuss how the mailers actually make delivery. Where ever delivering a message is mentioned, all that is meant is that the mailer should do whatever it needs to do to transfer a message to a remote site, given a domain name for that site. (For example, an SMTP mailer will try to get an address for the domain name, which involves another query to the domain system, and then, if it gets an address, connect to the SMTP TCP port). The mechanics of actually transferring the message over the network to the address associated with a given domain name is not within the scope of this memo.

It is possible that the list of MXs in the response to the query will be empty. This is a special case. If the list is empty, mailers should treat it as if it contained one RR, an MX RR with a preference value of 0, and a host name of REMOTE. (I.e., REMOTE is its only MX). In addition, the mailer should do no further processing on the list, but should attempt to deliver the message to REMOTE. The idea here is that if a domain fails to advertise any information about a particular name we will give it the benefit of the doubt and attempt delivery.

If the list is not empty, the mailer should remove irrelevant RR's from the list according to the following steps. Note that the order is significant.

For each MX, a WKS query should be issued to see if the domain name listed actually supports the mail service desired. MX RRs which list domain names which do not support the service should be discarded. This step is optional, but strongly encouraged.

> **NOTE:** Experience has shown that WKS is not widely supported, so the WKS step in MX processing **should not** be used.

If the domain name LOCAL is listed as an MX RR, all MX RRs with a preference value greater than or equal to that of LOCAL's must be discarded.

After removing irrelevant RRs, the list can again be empty. This is now an error condition and can occur in several ways. The simplest case is that the WKS queries have discovered that none of the hosts listed supports the mail service desired. The message is thus deemed undeliverable, though extremely persistent mail systems might want to try a delivery to REMOTE's address (if it exists) before returning the message. Another, more dangerous, possibility is that the domain system believes that LOCAL is handling message for REMOTE, but the mailer on LOCAL is not set up to handle mail for REMOTE. For example, if the domain system lists LOCAL as the only MX for REMOTE, LOCAL will delete all the entries in the list. But LOCAL is presumably querying the domain system because it didn't know what to do with a message addressed to REMOTE. Clearly something is wrong. How a mailer chooses to handle these situations is to some extent implementation dependent, and is thus left to the implementor's discretion.

If the list of MX RRs is not empty, the mailer should try to deliver the message to the MXs in order (lowest preference value tried first). The mailer is required to attempt delivery to the lowest valued MX. Implementors are encouraged to write mailers so that they try the MXs in order until one of the MXs accepts the message, or all the MXs have been tried. A somewhat less demanding system, in which a fixed number of MXs is tried, is also reasonable. Note that multiple MXs may have the same preference value. In this case, all

MXs at with a given value must be tried before any of a higher value are tried. In addition, in the special case in which there are several MXs with the lowest preference value, all of them should be tried before a message is deemed undeliverable.

Minor Special Issues

There are a couple of special issues left out of the preceding section because they complicated the discussion. They are treated here in no particular order.

Wildcard names, those containing the character '*' in them, may be used for mail routing. There are likely to be servers on the network which simply state that any mail to a domain is to be routed through a relay. For example, at the time that this RFC is being written, all mail to hosts in the domain IL is routed through `RELAY.CS.NET`. This is done by creating a wildcard RR, which states that *.IL has an MX of `RELAY.CS.NET`. This should be transparent to the mailer since the domain servers will hide this wildcard match. (If it matches `*.IL` with `HUJI.IL` for example, a domain server will return an RR containing `HUJI.IL`, not `*.IL`). If by some accident a mailer receives an RR with a wildcard domain name in its name or data section it should discard the RR.

Note that the algorithm to delete irrelevant RRs breaks if LOCAL has a alias and the alias is listed in the MX records for REMOTE. (E.g. REMOTE has an MX of ALIAS, where ALIAS has a CNAME of LOCAL). This can be avoided if aliases are never used in the data section of MX RRs.

Implementors should understand that the query and interpretation of the query is only performed for REMOTE. It is not repeated for the MX RRs listed for REMOTE. You cannot try to support more extravagant mail routing by building a chain of MXs. (E.g. `UNIX.BBN.COM` is an MX for `RELAY.CS.NET` and `RELAY.CS.NET` is an MX for all the hosts in .IL, but this does not mean that `UNIX.BBN.COM` accepts any responsibility for mail for `.IL`).

Finally, it should be noted that this is a standard for routing on the Internet. Mailers serving hosts which lie on multiple networks will presumably have to make some decisions about which network to route through. This decision making is outside the scope of this memo, although mailers may well use the domain system to help them decide. However, once a mailer decides to deliver a message via the Internet it must apply these rules to route the message.

Examples

To illustrate the discussion above, here are three examples of how mailers should route messages. All examples work with the following database:

```
A.EXAMPLE.ORG     IN     MX     10     A.EXAMPLE.ORG
A.EXAMPLE.ORG     IN     MX     15     B.EXAMPLE.ORG
A.EXAMPLE.ORG     IN     MX     20     C.EXAMPLE.ORG
A.EXAMPLE.ORG     IN     WKS    10.0.0.1     TCP     SMTP

B.EXAMPLE.ORG     IN     MX     0       B.EXAMPLE.ORG
B.EXAMPLE.ORG     IN     MX     10      C.EXAMPLE.ORG
B.EXAMPLE.ORG     IN     WKS    10.0.0.2     TCP     SMTP

C.EXAMPLE.ORG     IN     MX     0      C.EXAMPLE.ORG
C.EXAMPLE.ORG     IN     WKS    10.0.0.3     TCP     SMTP

D.EXAMPLE.ORG     IN     MX     0      D.EXAMPLE.ORG
D.EXAMPLE.ORG     IN     MX     0      C.EXAMPLE.ORG
D.EXAMPLE.ORG     IN     WKS    10.0.0.4     TCP     SMTP
```

In the first example, an SMTP mailer on D.EXAMPLE.ORG is trying to deliver a message addressed to A.EXAMPLE.ORG. From the answer to its query, it learns that A.EXAMPLE.ORG has three MX RRs. D.EXAMPLE.ORG s not one of the MX RRs and all three MXs support SMTP mail (determined from the WKS entries), so none of the MXs are eliminated. The mailer is obliged to try to deliver to A.EXAMPLE.ORG as the lowest valued MX. If it cannot reach A.EXAMPLE.ORG it can (but is not required to) try B.EXAMPLE.ORG. and if B.EXAMPLE.ORG is not responding, it can try C.EXAMPLE.ORG.

In the second example, the mailer is on B.EXAMPLE.ORG, and is again trying to deliver a message addressed to A.EXAMPLE.ORG. There are once again three MX RRs for A.EXAMPLE.ORG, but in this case the mailer must discard the RRs for itself and C.EXAMPLE.ORG (because the MX RR for C.EXAMPLE.ORG has a higher preference value than the RR for B.EXAMPLE.ORG). It is left only with the RR for A.EXAMPLE.ORG, and can only try delivery to A.EXAMPLE.ORG.

In the third example, consider a mailer on A.EXAMPLE.ORG trying to deliver a message to D.EXAMPLE.ORG. In this case there are only two MX RRs, both with the same preference value. Either MX will accept messages for D.EXAMPLE.ORG. The mailer should try one MX first (which one is up to the mailer, though D.EXAMPLE.ORG seems most reasonable), and if that delivery fails should try the other MX (e.g. C.EXAMPLE.ORG).