AmiTCP/IP

System Manual

`Revision: 2.4`

`amitcp-group@hut.fi`

Tomi Ollila Pekka Pessi Markus Peuhkuri Jarno Rajahalme

August 13, 1993

Foreword

What Is AmiTCP/IP

The Programming Project is a software engineering course here at Helsinki University of Technology. A group of 3—4 students designs, implements and tests a complete working software system during this course.

Our purpose was to provide a free *TCP/IP* implementation for all Amiga users. Lack of a standard networking interface was an irritating shortcoming of the Amiga systems.

AmiTCP/IP is a *protocol stack* implementing basic Internet protocols on top of any *SANA-II* network device driver (for *Ethernet*, *SLIP*, etc.). The protocol stack offers the standard Berkeley *Socket* application program interface (*API*) to the TCP/IP protocols implemented as an Amiga shared library.

Using AmiTCP/IP and appropriate client program you can connect to any services available on your TCP/IP network.

How Is This Achieved

At first we planned to write the whole protocol stack from scratch. After examining standards about TCP/IP and bibliography about its implementations we become aware of the possibility of using the BSD NET/2 code as a base of our work. We discussed about the idea and found good reasons why the BSD code should be used. Eventually we decided to take that path with the project.

AmiTCP/IP is built around *BSD NET/2* networking code. Because the networking part of the BSD *UNIX* kernel is rather isolated, only a few kernel routines had to be rewritten to suit the architecture of the AmiTCP/IP.

SANA-II and shared library interfaces were added to the bottom and top of the modified NET/2 code, respectively.

Why Is the BSD Code Used

Here are some of the pros of the decision:

- We don't need to reimplement the most complex part (and all the bugs) of the system.
- We noticed that the BSD kernel support for the networking part is easy enough to implement using powerful Amiga features.
- When improvements are done to the NET/2 code, we can improve AmiTCP/IP by simply replacing the old code with the changed one.
- We are offering the socket application interface. It is the most used interface in networking programs with TCP/IP. Using the NET/2 code makes it easy to implement the BSD socket related function call interface as our API.
- We can easily port crucial network utilities from the BSD.

About SANA-II

Bottom level interface to the SANA-II device drivers was an obvious choice. SANA-II is the Amiga standard interface for network devices, and most manufacturers are shipping SANA-II compatible device drivers for their hardware. There are also some freely distributable SANA-II drivers e.g. for the standard serial interface.

Although there may be few problems with different network types the AmiTCP/IP should be able to use all SANA-II device drivers which support IP.

Release 2.0

This release of AmiTCP/IP includes only the protocol stack (**AmiTCP**), test programs, basic network utilities and example client and servers. The distribution still lacks most basic client and server programs.

The second release of the AmiTCP/IP contains some improvements and bug fixes over the first release. It is incompatible with previous version in binary level, however. Old applications and the new library *DO NOT* work together. If you have an application compiled with previous version of AmiTCP/IP, it *MUST* be recompiled. We hope that this is the only downward incompatible release.

The AmiTCP/IP Group requests all users of the AmiTCP/IP to return any new applications or improved versions to the AmiTCP/IP Group, `<amitcp-group@hut.fi>`, and grant it the right to redistribute them.

Acknowledgments

We would like to thank

- Computer Systems Research Group from University of California, Berkeley, for developing BSD Unix system.
- Carnegie Mellon University for Mach bsdss port of BSD Net/2.
- Antti Louko <alo@hut.fi>, our project supervisor, from the Computing Centre of the Helsinki University of Technology.
- Jukka Partanen <jtp@cs.hut.fi> for providing the GNU CC cross-compiling environment for the Amiga in the HP 9000/700 workstations.
- Timo Rossi <trossi@jyu.fi> for providing the SANA-II drivers for Western Digital Ethernet cards used in testing.

Further on, Jarno and Markus would like to thank their wives Maarit and Katri for forgiving those late nights (or *early mornings*) that we spent with this project.

AmiTCP/IP contains material derived from BSD net/2 release. The following acknowledgement should be included in all AmiTCP/IP distributions:

Some code of the AmiTCP/IP originates from the BSDSS 4 by CMU, which is hereby acknowledged:

About This Manual

How This Manual Is Organized

We have tried to organize this manual so that you are not forced to read material that you are not interested in. In order to achieve this we have divided the text into following chapters:

1. *User's Manual* gives you basic information about using the AmiTCP/IP system. Descriptions about our debugging tools **qwriter** and **agnet.device** is also included here.

2. *Interfaces* describes briefly all external interfaces of the AmiTCP/IP.

3. *Programmer's Manual* gives you the needed information to use AmiTCP/IP to make your own network-aware programs.

4. *Internal Description* gives you information about the internal structure of the AmiTCP/IP-system. This is meant for all of you who are interested in the internal design of the AmiTCP/IP.

5. *Implementation Notes* is written for those of you who are going to maintain AmiTCP/IP or plan to develop it further.

There are also two appendices, the user and programmer reference guides. These appendices are in the AutoDoc format and they are also available in machine readable form.

If you have trouble understanding the jargon[1] or TLAs[1] refer to the glossary at the end of this manual.

Typographical Conventions

To help you to understand the text in this manual, following typefaces and punctuation are used:

Roman is the basic typeface used throughout the text.

Slanted is used in text which we want you to take special attention on.

Italics is used to introduce new terms and concepts which will be explained shortly. *Italics* is also used to refer to the names of the publications and to the sections of this manual.

[1] See glossary.

`Typed Text` is used for literal program code and items specific to the implementation, such as function names, identifiers and C language keywords.

Bold text is used to refer to specific files and command names used by AmiTCP/IP.

Syntax Conventions

There are few syntactic conventions used in this manual:

[...] Square brackets are used for bibliographical references, list of which can be found from the end of this manual.

⟨*argument*⟩ User supplied required arguments are printed between angle brackets.

[*argument*] User supplied optional arguments are printed between square brackets.

`function()` C function names are followed by a pair of parenthesis. The preprocessor macros—as opposed to the genuine functions—are in upper case.

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

1. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

2. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

3. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   (a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   (b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

(c) if the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

4. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   (a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

5. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

6. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

7. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

8. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as

to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

9. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

10. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

    Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

11. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software

which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

12. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

13. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Contents

Chapter 1

User's Manual

1.1 Installation

AmiTCP/IP installation is made by standard Commodore Amiga Installer which provides an easy installation. Before you can proceed, you should acquire some information, list of which follows.

Your should ask your local network administrator for few things to be able to connect your Amiga to a network. Take this page or copy of it with you to make sure that you don't forget anything:

1. Internet protocol address allocated for your computer. This is a series of four decimal numbers separated by dots. For example, "`130.233.161.40`" is a legal internet address.

2. Destination IP address of a point–to–point link. This applies only if you use a point–to–point medium, such as a serial line.

3. Netmask for your network[1] This determines which part of your internet address is used to specify the network. Netmask is like "`255.255.255.0`". If you do not supply this value, a default computed from the IP address is used.

4. Name of your host. This is a mnemonic name which can usually be used instead of the internet address to refer to your computer. This is like "`Amiga1`".

5. Domain name of your site. For example, "`hut.fi`" is the domain name for the Helsinki University of Technology in Finland.

[1]Netmask is usually needed only if you are connected to several networks or a network capable to broadcasts.

6. Address(es) of the Domain Name Server(s) which will be used to translate host names to corresponding internet addresses. A local host database must be used instead if there is no name server on your domain.

7. Address of the default gateway via which to send packets to the hosts not on your local network. There may be more than one of them.

8. The network interface to use. This may be Ethernet, SLIP or something else. We assume that you have proper hardware installed and the appropriate Sana-II compatible device driver installed in directory **Devs:networks**. AmiTCP/IP needs the name of the device driver in order to use it. The unit number for the device in question is also needed (usually this is `0`).

Installation starts by double-clicking the **Install_AmiTCP-2.0**-icon of the distribution disc. You can select the level which specifies how many choices are left to you by the installation script. After a successful installation your computer is ready to communicate with the network. See the file **README.FIRST** for more information about the installation procedure.

If you plan to compile applications which use AmiTCP/IP, you should add two AmigaDOS assigns for the compiler to find appropriate files. Assign the name `NETINCLUDE:` to the **netinclude** directory of the AmiTCP/IP and the name `NETLIB:` to the directory **netlib**.

1.1.1 Using the SLIP Drivers

Some Commodore-Amiga Networking Group supplied SANA-II drivers are distributed with the AmiTCP/IP. The SLIP drivers included are taken from Olaf Seibert's distribution and has some bug fixes over the version 37.3. See the file **doc/slip.doc** for installing instructions for these drivers.

The version number of the included **slip.device** is 37.4. You can use this or newer SLIP driver with AmiTCP/IP with certain precautions. The included CSLIP device is similar to the SLIP device, it only uses the Van Jacobson header compression (see [Jacobson 1990] for description of this compression method). If your network connection supports CSLIP, you can use **cslip.device** instead of **slip.device** in all examples given in this section.

Used serial interface, baud rate and other parameters for the unit 0 of the SLIP device are specified in the configuration file **ENV:Sana2/slip0.config**. You must edit this file according your system setup before you can use the slip device. Copy the edited file to both **ENVARC:Sana2** and **ENV:Sana2** directories.

The AmiTCP/IP opens the slip device driver when given a following **ifconfig**[2] command:

[2] The **ifconfig** and **route** commands like described here are normally placed to the **startnet** script during AmiTCP/IP installation.

```
ifconfig slip.device/0 local-address remote-address
```

The `local-address` is the IP address (or name) given to your host, the `remote-address` is the IP address (or name) of the host you are connecting to with the serial line. You should also add a routing entry with the **route** command, for example

```
route add default remote-address
```

You can test your SLIP connection immediately with the command

```
ping remote-address
```

Common Problems with the SLIP Driver And AmiTCP/IP

The **ifconfig** command gives you the following error message, if there is some problems opening the interface:

```
ifconfig: ioctl (SIOCGIFFLAGS): no such interface
```

If the device driver is not installed properly or there is no configuration file, AmiTCP/IP gives the following log message:

```
<3> OpenDevice: Device or unit failed to open
```

In this case, you should check that the file **ENV:Sana2/slip0.config** actually exists and the device driver is in the correct directory.

If the device driver is ok, but the needed serial device is busy (or you have set the slip driver to offline state), AmiTCP/IP logs the following message:

```
<3> slip.device: Device driver is offline (Unit is currently offline)
```

Then you must free the serial interface, set the slip device to the online state and give the previous **ifconfig** command with the "`up`" option[3] again.

If AmiTCP/IP gives no error messages and the serial link still does not work, it may be due to incorrect information in the **ENV:Sana2/slip0.config**-file. Check that the baud rate is correct. Note that the slip device loads the configuration file only on startup, which means that you must restart the AmiTCP/IP if you have already configured the slip driver with an `ifconfig` command.

[3]The `up` is needed because the AmiTCP/IP implicitly adds this option when an address to the interface is *first* given. If the second **ifconfig** specifies the same address as before, the `up` option must be given explicitly.

1.2 Configuration

AmiTCP/IP is normally started from the command line with a startup script. It starts the AmiTCP/IP process, which loads the configuration files. The script also contains **ifconfig** and **route** commands to open the SANA-II device drivers and add the needed route entries, respectively.

The AmiTCP/IP can be configured with both option files and ARexx commands. The option files are stored in the text format and they can be edited with any text editor. The '`#`' character starts a comment, which continues to the end of the line. All configuration commands are presented in the section 2.4 starting from the page 19.

Configuration files for AmiTCP/IP are stored in the directory **AmiTCP:db**. It contains at least following files:

AmiTCP:db/AmiTCP.config AmiTCP/IP internal configuration.

AmiTCP:db/netdb Network database information, ie. host names, networks, services, protocols, domain name servers and domains.

AmiTCP:db/hosts.equiv Host equivalence data used by **rlogind**, **rshd** etc. (these daemons are currently unimplemented).

AmiTCP:db/inetd.conf Services provided by **Inetd**, the internet super server.

1.2.1 Options

Normally AmiTCP/IP reads options from the file **AmiTCP:db/AmiTCP.config**. You can specify an option file also on the command line (eg. `AmiTCP WITH myopts`) or with the tool type `WITH`. You can prevent AmiTCP/IP from loading the default option file by the `NOCONFIG` command line option.

1.2.2 Network Database

The file **AmiTCP:db/netdb** is used to store the local net and host name information. There is an example file in the distribution, which you must change according to your local network environment. The syntax for the network database file is same as the syntax for the Arexx command `add` (see page 22).

The most important entries are the name of your own host and the important hosts in your local net. For instance, if your host name is `amiga1.hut.fi` and your internet address (IP number) is `130.233.61.50`, then there is probably a following line in your **AmiTCP:db/netdb** file:

```
HOST 130.233.61.50 amiga1.hut.fi amiga1 amy1
```

The first entry after the keyword `HOST` is the internet address of your host. It is followed by the official name and an optional list of alias names to your host.

Normally, all host and network names will be translated to the actual internet addresses by a name server. If you can not use the name service, then all host names must be in the **AmiTCP:db/netdb** file or in an included file. In that case it may be easiest to copy the **/etc/hosts** file from a nearby Unix machine.

You can add host names to the network database with ARexx commands at any time. For instance, the following command adds a host called `vipunen`:

```
rx "address AMITCP add host 130.233.224.20 vipunen.hut.fi vipunen"
```

There are also service and protocol entries in the network database files. Do not change them if you are not an expert user.

1.3 Internet Addressing

Each host in the Internet must have an unique 32 bit binary address. This address identifies the location of host in the internet in an unambiguous way. The internet address is divided into a net part and a host part. Because of this elegant feature a single address identifies a unique host as well as the network which the computer is connected to.

When selecting a route to a particular host the whole address is considered first. If no direct route to the host can be found, a route to the host's network can be considered. If no route to either host or net can be found, the packet is sent to the default gateway.

Because the the internet address defines also the network which the computer is connected, they must actually be given one per network connection. If a host is directly connected to several networks, it has also several distinct internet addresses.

1.3.1 Internet Dot Notation

Since it is rather inconvenient to use 32-bit integers, the internet address is normally written in the *Internet dot notation.* It is a series of decimal numbers separated by dots.

The one with four separate octets is the most common form of the notation. See the reference for the function **bsdsocket.library/inet_addr()** to additional information (at Appendix B.3.1, page 186).

Format	Size	Example
a	32-bit	169510179
a.b	8-bit.24-bit	10.1738019
a.b.c	8-bit.8-bit.16-bit	10.26.34083
a.b.c.d	8-bit.8-bit.8-bit.8-bit	10.26.133.35

Table 1.1: Four different forms for an Internet address.

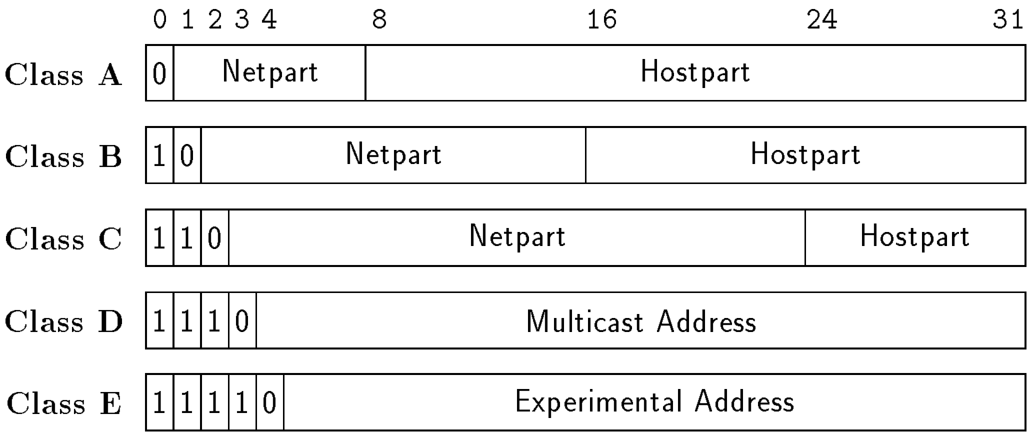

Figure 1.1: The Internet address classes. The different address classes can be identified by few first bits.

1.3.2 Nets and Routing

There are five classes of internet addresses in use, namely A, B, C, D and E classes. The difference between address classes is the length of the network and host part. An A class address has 7 bits for net address and 24 bits for host address, a B class address has 14 bits for net and 16 bits for host and a C class address has 21 bits for net and 8 bits for host. The D class is used for the IP multicast addresses and the E class for experimental use only[4].

The address class can be determined by the first octet of the address. If it is in the range 0—127, the address belongs to the A class. If it is 128–191, the address belongs to the B class. If the first octet is 192–223, the address belongs to the C class, otherwise it is an experimental or a multicast address.

[4]Routers not taking part to such experiments normally discard all packets of this class.

1.3.3 Subnets

The network addresses are managed by a central authority. Local administration manages the host addresses. Because network addresses are used in routing, the central authority is needed when the local network changes its structure. This might be quite a burden to the central authority, but there is a solution developed for the Internet. A network may be split into several subnets, which can then be managed by independent agencies. The central network management doesn't have to know anything about subnets and hosts in them.

For instance Kone corporation has been assigned a B class network 138.249.0.0. This network has been divided to C class like subnets, which are then given to the departments. The HAT department is given subnet 138.249.2.0, the HATMEC department subnet 138.249.5.0 and so on.

		0	8	16	24	
	Address	10001010	11111001	00000010	00011110	138.249. 2. 30
AND	Netmask	11111111	11111111	11111111	00000000	255.255.255. 0
=	Subnet	10001010	11111001	00000010	00000000	138.249. 2. 0

Figure 1.2: Using netmask to find out subnetwork address.

The subnet division is controlled by the *netmask* parameter. The actual subnetwork address is obtained by bit-wise `AND` between the netmask and the internet address. This subnet address is used in routing just like a normal net address. However, only hosts directly connected to the divided net have to know anything about netmasks and subnets.

1.3.4 Broadcast Addresses

There are two kind of broadcast addresses in the Internet. An obsolete form used in older BSD 4 versions used internet address with host part 0 as the broadcast address. (The newer software usually considers addresses like those as network addresses.) According the the current Internet standard the broadcast addresses have the host part all 1's.

1.4 Utilities

Currently, only a few networking utilities are ported for the AmiTCP/IP. They are used to monitor and control network operation with AmiTCP/IP. There is a separate reference entry for each network utility and daemon in the document file **netutil.doc**. This autodoc file is included in the appendix A, starting from page 127.

Interface Address	Netmask	Network Address	Broadcast Address
10.14.123.2	255.255.255.0	10.14.123.0	10.14.123.255
130.244.5.4	255.255.255.128	130.244.5.0	130.244.5.127
192.36.148.21	255.255.255.0	192.36.148.0	192.36.148.255

Table 1.2: Examples of netmasks, network addresses and broadcast addresses.

1.4.1 arp

arp displays and modifies the internet to hardware address translation tables used by the *Address Resolution Protocol*. It is used if you are connected to a broadcast network, eg. Ethernet or Arcnet. You may examine, add, and delete ARP entries. Usually **arp** is used to find out hardware address of hosts.

The hardware addresses are given as hexadecimal strings, each octet separated by a colon.

Examples:

```
1.> arp -s puucee 8:0:9:32:f2:4c pub
```

This command sets the hardware address for the host 'puucee' as '`8:0:9:32:f2:4c`'.

1.4.2 ifconfig

ifconfig configures network interface parameters. It assigns an internet address to a SANA-II network interface or sets interface parameters. **ifconfig** is normally used at the AmiTCP/IP startup.

Examples:

```
1.> ifconfig lo/0 127.1
```

This command marks internal loopback device up, and attaches the internet address 127.0.0.1 to it.

```
1.> ifconfig cslip.device/0 inet 193.102.4.144 193.102.4.129
```

This command starts the `CSLIP` driver, attaches an address 193.102.4.144 (our internet address) and a destination address 193.102.4.129 (the internet address of the host you are connecting) to it.

```
1.> ifconfig a2065.device/0 inet 193.124.100.66 +
    netmask 255.255.255.192 up -arp
```

This command loads an ethernet driver for the Commodore A2065 Ethernet adapter unit 0, marks it `UP`, disables ARP protocol for it, attaches an inet address 193.124.100.66 to it and sets its netmask to 255.255.255.192. A bitwise logical `AND` of the netmask and the address for the interface forms the *subnet address*, in this case 193.124.100.66. All packets aimed to hosts with same subnet address (that is hosts 193.124.100.65–193.124.100.126) are routed to this interface.

1.4.3 letnet

letnet is a simple TCP connection tool. With it you can connect to a TCP port at given host. Letnet reads data from standard input and sends it to the host. Likewise it receives data from the connection and writes it to the standard output. Letnet terminates upon shutdown of the socket or receiving `SIGBREAKF_CTRL_C` signal. Letnet does not handle telnet protocol.

1.4.4 netstat

netstat shows network status. It symbolically displays information of network data structures of the AMITCP/IP. There are several standard AmigaDOS style options:

By default displays all sockets that are not listening.

ALL displays all sockets.

STATUS displays values of several protocol-related variables.

Example:

```
1.> netstat status
```

Netstat is an ARexx script which uses the **AMITCP** port of the AMITCP/IP to collect the information. See section 2.5.1 (p. 20) for details about the ARexx commands supported by the AMITCP/IP.

1.4.5 NapsaTerm

NapsaTerm is a VT102 terminal emulator for logging remotely to Unix hosts with *rlogin* protocol. It is based on the **NiftyTerm** version 1.2 for Amiga by Chris Newman and William Todd.

```
1.> run napsaterm -d net happi.hut.fi
```

This command starts VT102 emulator with a rlogin connection to the host `happi.hut.fi`.

1.4.6 offline

offline[5] sends the `S2_OFFLINE` command to the given SANA-II device driver. This command is normally used to disconnect SANA-II device driver from the network adapter hardware. After this command the device driver does not accept any more read or write requests.

Example:

```
1.> offline slip.device/0
```

This command puts the SLIP driver unit `0` offline, which frees then the serial port to other use.

1.4.7 online

online sends the `S2_ONLINE` command to the given SANA-II device driver. The device driver restarts the network adapter hardware and accepts read and write request again.

Example:

```
1.> online a2066.device/0
```

This command tries to start the Commodore ARCNET network adapter.

1.4.8 route

route manipulates the routing tables. It is normally used at the AmiTCP/IP startup sequence to provide the default routing information in the routing tables. All needed routing information can be provided with this command.

Routes determine into which network IP packets are sent. Usually there is only need to set the default route to your gateway and the route to the host itself through the loopback device. More route entries may be needed if your host is used as a gateway.

1.5 Errors and Logging

There are two kinds of error reports AmiTCP/IP produces:

Errors might be produced when a user process or a network supplies invalid data. These don't cause any problems to AmiTCP/IP. These events are logged[6] to help the user to find out possible problems with his/her system.

[5] **offline** and **online** are compatible with the Commodore supplied SANA-II utilities with same names. However, they support the AmiTCP/IP naming convention.

[6] if logging is not disabled

Failures are more serious. These mean that AmiTCP/IP has entered to state where there is no way out.

In Unix systems these cases lead to `panic()` and immediate restart. This is because in Unix systems networking software is integral part of the kernel and if something goes wrong, it usually means that the whole kernel is corrupted. This is not the case in Amiga and AmiTCP/IP, however. As AmiTCP/IP runs as a normal process, rest of the system can work even if networking fails.

In Unix systems the `syslogd` takes care of the error reporting using an attached console. In AmigaOS there is no general failure reporting system, only User Requesters which are inadequate for non-interactive use. As logging could be used even while in interrupt level, AmiTCP/IP can't write to log-files itself. This is why the actual logging is done by the NETTRACE process. It is started at AmiTCP/IP startup and can be configured through `AMITCP` ARexx port.

ARexx commands can be found in section 2.5.1 (p. 20).

1.6 Troubleshooting

This section is reserved for the frequently asked questions and answers to them.

1.7 Testing Utilities

There are two testing utilities provided with the AmiTCP/IP. The **agnet**[7] controls the loopback pseudo device which simulates different networks. The **qwriter** simulates some typical application programs. They can be used together to test the integrity of the protocol stack.

1.7.1 Agnet

SANA-II pseudo device driver **agnet.device** works as a network between its units. Packets from one unit are sent to another according given destination address. You can also arrange an unit-to-unit connection, if you want to simulate point-to-point devices.

The loopback device provides nearly full SANA-II device driver functionality. It provides different hardware types, packet types and the variable hardware address width from 0 to 128 bits. It collects all the required statistics. It does *not* provide the multicast functionality.

[7] As Good NETwork as you like

A special command file **env:sana2/agnet*n*.config** is used to configure and control the device. Configuration parameters may be specified by supplying its keyword and the wanted value. Any parameter may be left out.

The configuration file parsing template is
`WIRE/K,MTU/N/K,MINTU/N/K,BPS/N/K,ADDR=ADDRESS/K,`
`DELAY/N/K,DEV=DEVIATION/N/K,ERRORS/K/N,LOSS/K/N,DST=DSTUNIT/K/N`.[8]

User configurable parameters are:

`WIRE` ⟨*wiretype*⟩ is the hardware type which device driver will report. Other default parameters are set according to it. Default hardware type is Ethernet. With different wire types you can test for instance Ethernet specific features of AmiTCP/IP without killing the whole Ethernet segment of your network.

`ADDRESS` ⟨*hardware address*⟩ is the hardware addresses for the unit. Address is specified as a hexadecimal octet string. Octets are separated by colons. The length of the address depends of the hardware type. For example, `8:0:9:14:74:53` is a legal Ethernet hardware address. *Alias:* `ADDR`.

`DSTUNIT` ⟨*unit number*⟩ is the destination unit for a unit-to-unit connection. If this parameter is given, all packets sent to the unit are relayed to specified destination unit regardless of the packet destination address.

`MTU` ⟨*number of octets*⟩ is the maximum packet size that pseudo network will accept. Default value is 1500 *octets*, same as the MTU for the Ethernet.

`MINMTU` ⟨*number of octets*⟩ is the minimum packet size that pseudo network will accept. Smaller packets are padded to this minimum length. Default value depends on the hardware. It is for instance 46 octets for the Ethernet.

`BPS` ⟨*transmission speed*⟩ is the bit speed which device driver will report to the user.

`DELAY` ⟨*delay time*⟩ specifies the delay of network in milliseconds. By default there is no delay.

`DEVIATION` ⟨*standard deviation*⟩ specifies delay variance in milliseconds. Packets are delayed randomly which means that packets may arrive in different order they were sent. Deviation without delay is meaningless. By default the variance is 0 ms and all packets have same delay.

[8] Alternative forms of keywords are specified by using `alt=keyword`. Modifier `/N` specifies that the parameter is numeric.

`ERRORS` ⟨*probability*⟩ is the probability of bit errors. This value can be in range [0..100 000 000], ERRORS/100 000 000 indicating probability of a single bit error. Error rate 100 000 000 means that all bits have random value. By default there are no errors (value 0).

`LOSS` ⟨*probability*⟩ defines the probability of the packet loss. This is a number in the range [0..1 000 000], LOSS/1 000 000 indicating probability. By default there is no packet loss. Note that value 1 000 000 means that all packets are lost.

ARexx Port for `agnet.device`

The Arexx port is normally named as `AGNET1`. If there is multiple instances of the **agnet.device**, the second instance has the port `AGNET2` and so on.

There is currently three different Arexx commands:

`UNIT` *Alias:* `U`

Some configuration parameters may be changed at run-time by Arexx commands. The Arexx configuration commands follow the same syntax as the configuration file. The user must specify the unit to configure by preceding the command with the keyword (`UNIT`, abbreviated as `U`) and a decimal unit number. For instance, the following is a legal Arexx configuration command:

```
address AGNET1 unit 0 bps 2400 wire slip delay 100
```

`QUERY` *Alias:* `Q`

The current configuration may be queried from the device by an ARexx command `Q=QUERY`. The command keyword is followed by an unit number and wanted configuration parameters. You can also give a keyword `ALL`, if you want all configuration parameters. For example, following command prints wiretype, delay, deviation and loss probability:

```
options results
address AGNET1 query 0 wire delay deviation loss
say result
```

`EXIT` *Aliases:* `E, EXPUNGE`

The device may be expunged from the system memory by this Arexx command. The command fails, if there are some currently opened units.

1.7.2 Qwriter

qwriter[9] works on the application layer. It is run as two distinct processes. A process acts as a server or a client. It creates one socket and sends and receives data through the socket. It is able to control data consistency and measures both latency times and throughput (if appropriate).

Server part is started first in all tests. Server starts to listen to a port, and you can start the client. Tests are configurable, basic test types are described below.

telnet simulates typical interactive use.

In this test a TCP connection is created. The client end sends one character at a time and the server replies with four or more characters at a time.

The most important measurement is the latency time between the sent character and the received echo.

nfs simulates the data transfer of a *network file system*.

In this test UDP messages are sent between processes.

Both latency and the throughput are important measurements in this test.

ftp is a simple TCP connection.

In this test a TCP connection is created, then some data is transferred from one end to the another.

In this test throughput is the key measurement.

Command Line Options

-s Act as a server (default).

-c Act as a client.

-t ⟨*type*⟩ Describes type of test: **ftp**, **nfs** or **telnet**. Type can be abbreviated to shortest unique length. Default is **telnet**.

-n ⟨*count*⟩ Describes how many times packets are sent back and forth. Applies to nfs and telnet. Default for ⟨*count*⟩ is 1.

-l ⟨*length*⟩ Describes length of data. Default is 1000 *octets*. Length has different meaning in different test contexts:

[9] Quick WRITER

ftp Length of the requested data for the client. Length parameter is meaningless for a "ftp" server.

nfs The size of the packet to be sent to the other end.

telnet The maximum length of data returned from the server. Actual length is random value between 4 and *length* octets. See also option "-d". This parameter is meaningless for a client.

-d ⟨*n1,n2,...,nN*⟩ This option has meaning for the telnet server and overrides the "-l" option. It specifies distibution of data lengths of replied packets to help better simulate typical interactive use where replies are usually short, but sometimes long (few kilobytes).

Only $k = 2^i \leq N$ first numbers have effect. Probability to have length less than nj $(1 \leq j \leq k)$ is approximately

$$P(j) = P(j-1) + \frac{1}{k}\sum_{l=j}^{k}\frac{nj}{nl}$$

$(P(0) = 0)$ This can vary depending on random numeber generator and because of discret nature of computing. The largest length is nk.

-b ⟨*size*⟩ Size of the buffer to be allocated for sends and/or replies. Default is 30 kilobytes. Amount of data to be send at once, if smaller than data to be sent. Maximum amount of data to be expected from the other end.

-p ⟨*port*⟩ the server port number. The default is 1500.

-h ⟨*host*⟩ Name of the host to connect. Has no meaning for servers. The default is address "127.0.0.1".

-q Calculate checksum and verify returned messages. This provides way to check for integraty of transferred data. Checksum method is FCS as described in PPP RFC (1331).

-T ⟨*timeout*⟩ Specifies timeout (in seconds) for retransmission. Only meaningful for nfs client. The default is 1.0 seconds.

Chapter 2

Interfaces

AmiTCP/IP provides Berkeley sockets compatible interface to the application programs. Networking applications can use this interface to use TCP and UDP protocols, for example.

AmiTCP/IP networking system uses SANA-II network devices. SANA-II is an Amiga standard network device driver interface at the *data link layer*.

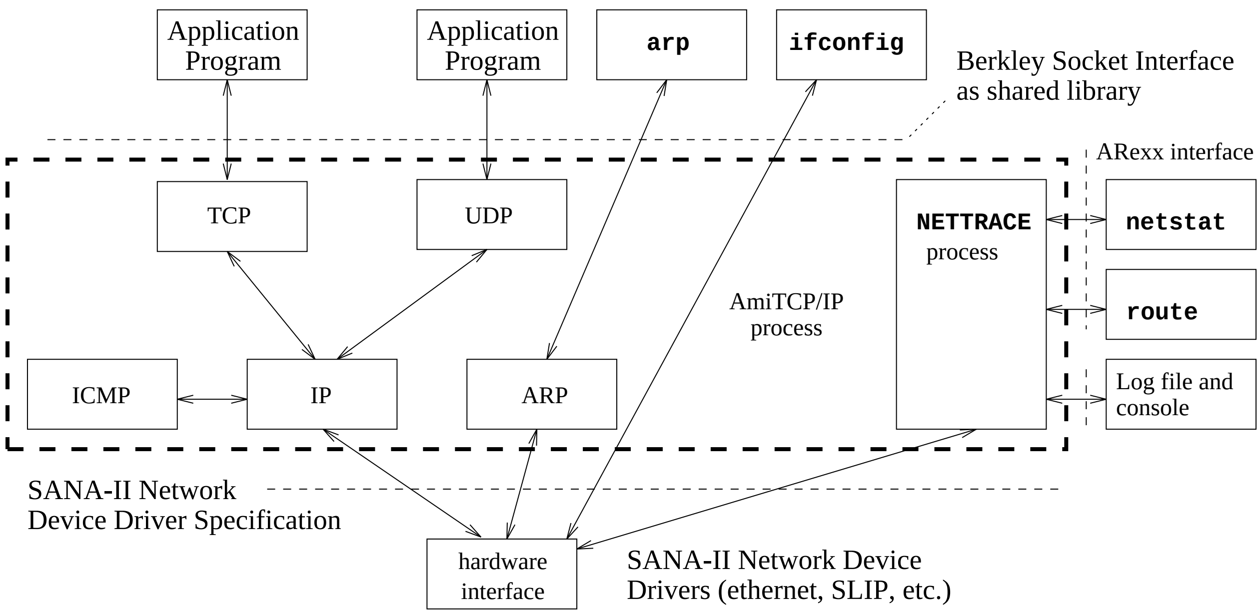

Figure 2.1: Interfaces between different modules of the AmiTCP/IP internetworking system.

There is also a standard ARexx interface for logging, statistics and configuration in AmiTCP/IP.

2.1 AmiTCP/IP Processes

AmiTCP/IP uses one protocol process. The protocol process starts listening specified SANA-II device drivers for incoming network *frames* and sends timeout requests periodically to the timer device. Output to the network is usually done in the API callers context concurrently with the input processing.

There is one other task called `NETTRACE` taking care of error reporting and the ARexx port of the AmiTCP/IP. The commands for this port are described later in the section 2.5.1.

2.2 Application Interface

Application Interface is implemented as a standard Amiga shared library. Each task that opens the library (using the `OpenLibrary()` call) is given an unique data area (called a library base), which AmiTCP/IP uses it to store task specific information (see section 3.1.2 for more on this subject). Once the library is opened, applications can call the standard socket functions such as `socket()`, `connect()`, `bind()`, `listen()`, `accept()`, `recv()` and `send()`.

2.3 SANA-II Interface

AmiTCP/IP can use (hopefully) any SANA-II [SANA-II 1992] device. However, there may be problems with packet types and addressing.

Many protocols may use the same hardware if each protocol uses unique packet type number. For instance, in the Ethernet the IP protocol have packet type 2048, the Xerox NS protocol uses the packet type 1536. The device driver directs packets to different protocols depending of their type. IP packet type varies in different hardware, for instance *ARCNET* uses packet type 240 for IP[1].

SANA-II devices are required to follow the hardware type numbers assigned in the [Reynolds 1990]. There is only one hardware type which for instance all Ethernet drivers should use. The SANA-II interface module uses the hardware type number to determine packet types for each protocol.

The hardware level addressing complicates the situation further. It is not possible to provide a general addressing scheme in hardware level. AmiTCP/IP can use a driver only if there is some mapping from IP addresses to hardware addresses.

[1]There is an extension to ARCNET IP that is incompatible with old one. It uses packet type 212 for IP.

Sometimes SANA-II network driver uses an IP compatible addressing scheme, for instance SLIP or *PPP* drivers[2] use an IP address as the hardware address. Other solution is to use Address Resolution Protocol (*ARP*). With ARP AmiTCP/IP can map any IP address to the corresponding hardware address.

For installing appropriate SANA-II device driver the manual of the device driver should be consulted. Note that if the device driver is not installed into the **devs:networks** directory, a full path must be specified when the driver is first referred to after the startup.

A network device driver can be attached to the AmiTCP/IP with **ifconfig** command. This can be done manually or by the network startup script.

2.3.1 Interface `ioctls`

The AmiTCP/IP provides an upward compatible `ioctl` interface to configure the network interfaces. Refer to the autodoc entry **protocols/if** for standard BSD `ioctl` interface (see appendix D.1.3, page 230). The extension for this interface is meant to configure hardware dependent parameters in an uniform way.

The ARP interface have been modified to be slightly more generic. The length of the hardware address is now passed in the `arp_ha` address field. A new `ioctl` accesses the contents of the whole ARP mapping cache. (In the BSD the static ARP table is accessed via the **/dev/kmem**.)

2.4 Configuration Files

The initial state of AmiTCP/IP is set by the configuration files. The syntax differs from one configuration file to another. For each file format there is usual a corresponding ARexx command with the same syntax. For instance, configuration options, which are set in the file **AmiTCP:db/AmiTCP.config**, may be set with the `SET` command by an ARexx script. The entries of the **AmiTCP:db/netdb** file can be added manually with the `ADD` command.

2.5 ARexx Interface

In Amiga environment Exec message ports are standard way to communicate between applications. Because of the ease of programming and the efficiency the ARexx programming language has become the standard in the Amiga and structure it uses for messages is also standard. AmiTCP/IP provides ARexx port named `AMITCP` which can be used to both

[2] At the hardware level these protocols do not use addresses at all.

configure and gather information from networking. This is much more elegant compared to the BSD Unix way to read directly from the kernel memory.

The AMITCP port can be used to set and query various variables and statistic tables. Brief example of the usage of the port follows:

```
/*c:rexx
 *
 * An example to query some information form AmiTCP/IP
 */

address AMITCP
options results /* get results back */

'Q' TASKNAME
say "Name of amitcp task is" result
'SET' TASKNAME "INET"
'QUERY' ICMP CHKSUM IP CHKSUM TCP CONNECT UDP CHKSUM TASKNAME
/* Query count of ICMP, IP and UDP checksums and TCP connects */

parse value result with icps_checksum ips_badsum tcps_connects udps_badsum taskname
/* Parse values from result. (Names of the variables are from
 * the stat-structures
 */

say "New name is" taskname

say icps_checksum "bad ICMP checksums"
say ips_badsum "bad IP header checksums"
say tcps_connects "TCP connections established (including accepts)"
say udps_badsum "bad UDP checksums"
```

This example will produce output similar to following:

```
Name of amitcp task is AmiTCP
New name is INET
0 bad ICMP checksums
0 bad IP header checksums
2531 TCP connections established (including accepts)
0 bad UDP checksums
```

2.5.1 Commands

This section summarizes the commands recognized by the ARexx port of the AmiTCP/IP. The command lines may contain comments, which are started by the semicolon.

`ADD` ⟨*entry*⟩ This command adds entries to the network database. Templates for the entries are:

`WITH` ⟨*file name*⟩ [*PREFIX=*⟨*entry type*⟩] Include the file ⟨*file name*⟩. Entry type for the file may be specified with an optional prefix, which may be any of the entry types listed in this list. The file will be searched first from the **AmiTCP:db** directory, so that the actual path may be omitted, if the file resides in that directory.
For exmaple,

```
WITH hosts PREFIX=HOST
```

includes the file **hosts**, which contains only host entries. The host keyword must not be present at the file itself.

The `WITH` command is useful for including database files downloaded from Unix machines.

`H=HOST` ⟨*address*⟩ ⟨*name*⟩ [*aliases*] Add a host entry to the network database. Either `H` or `HOST` can be used as the keyword. The ⟨*address*⟩ is the internet address of the host, the ⟨*name*⟩ is the official name and [*aliases*] is an optional list of alias names for the host.
Example:

```
H 128.214.6.100 nic.funet.fi nic
```

`N=NET` ⟨*name*⟩ ⟨*address*⟩ [*aliases*] Add a net entry to the network database. Arguments are as above for the `HOST`.
Example:

```
N loopback-net 127 software-loopback-net
```

`S=SERVICE` ⟨*name*⟩ ⟨*port*⟩/⟨*protocol*⟩ [*aliases*] Add a service entry to the network database. The ⟨*port*⟩ is the port number for the ⟨*protocol*⟩ for which the server for the service will listen for the service requests. [*aliases*] is as above.
Example:

```
S daytime 13/tcp
```

`P=PROTOCOL` ⟨*name*⟩ ⟨*protocol number*⟩ [*aliases*] Add a protocol entry to the network database.
Example:

```
P tcp 6
```

`NS=NAMESERVER` ⟨*address*⟩ Add a name server entry to the network database. The ⟨*address*⟩ is the address of the name server in the internet dot-notation. The name servers are searched in the order specified in the database.
Example:

```
NS 130.233.224.1 ; santra.hut.fi
```

Note that the symbolic name is after the ';' character, which marks the start of an end of line comment.

`DO=DOMAIN` ⟨*domain name*⟩ Add a domain name entry to the network database. The ⟨*domain name*⟩ is the part following the first '.' in the official host name. The domain entries specify the domains from which a host name is searched. The search is done in the order of the entries added to the network database.
Example:

```
DOMAIN cs.hut.fi
```

`QUERY` ⟨*variable name*⟩ [...] Get the value of the variable. Many variables have a two-level hierarchical structure (especially for variables related to the statistics) in which both the name of the table and the name of the variable itself must be supplied.

Values for many variables can be queried with one command by putting names of the variables in a row.

`RESET` Reread the network database file **AmiTCP:db/netdb**.

`SET` ⟨*variable name*⟩ ⟨*variable value*⟩ [...] Set the value of a variable. Name of the variable is given as with `QUERY` and the value is given after the name.

Multiple variables can be set with one command by supplying name-variable pairs.

Note that some variables are read-only and others writeable only during configuration.

2.5.2 Variables

Here is a complete list of all configuration and network statistic variables.

`WITH` Include a file with multiple settings. (This pseudo variable is a extension to the `SET` command.)

`ICMP` Variables related to Internet Message Control Protocol. *Alias:* `IC`

`ERROR` Number of calls to icmp_error(). *Alias:* `E`

`SHORTOLD` No error because old IP was too short. *Alias:* `S`

- `ICMPOLD` No error because old was ICMP. *Alias:* `I`
- `CODE` ICMP code out of range. *Alias:* `CO`
- `TOOSHORT` Packet too short. *Alias:* `T`
- `CHKSUM` Checksum error. *Alias:* `CH`
- `LENGTH` Data length larger than packet. *Alias:* `L`
- `RESPONSES` Number of responses. *Alias:* `R`

`ICMPHIST` ICMP packet send and reception history. See page 28 for details on the output format. *Alias:* `ICH`

`IP` Variables related to Internet Protocol.

- `TOTAL` Total number of packets. *Alias:* `T`
- `CHKSUM` Checksum error. *Alias:* `CH`
- `TOOSHORT` Packet too short. *Alias:* `TOOSH`
- `TOOSMALL` Not enough data. *Alias:* `TOOSM`
- `HEADER` IP header length less than data size. *Alias:* `H`
- `LENGTH` IP header length larger than packet. *Alias:* `LE`
- `FRAGMENTS` Packet fragments received. *Alias:* `FS`
- `FDROP` Fragments dropped (duplicates, out of space). *Alias:* `FD`
- `FTIMEOUT` Fragments timed out. *Alias:* `FT`
- `FORWARD` Packets forwarded. *Alias:* `FO`
- `FWDCANT` Packets received for unreachable destination. *Alias:* `FW`
- `REDIRECTSENT` Packets forwarded on same net. *Alias:* `RED`
- `NOPROTO` Unknown or unsupported protocol. *Alias:* `N`
- `DELIVER` Packets consumed here. *Alias:* `D`
- `LOCALOUT` Total IP packets generated here. *Alias:* `LO`
- `ODROPPED` Lost packets due to nobufs, etc. *Alias:* `OD`
- `REASSEMBLED` Total packets reassembled ok. *Alias:* `REA`
- `FED` Output packets fragmented ok. *Alias:* `FE`
- `OFRAGMENTS` Output fragments created. *Alias:* `OF`
- `FCANT` Don't fragment flag was set, etc. *Alias:* `FC`

`TCP` Variables related to Transmission Control Protocol. *Alias:* `T`

- `CATTEM` Connections initiated. *Alias:* `CA`
- `ACCEPTS` Connections accepted. *Alias:* `A`
- `CONNECT` Connections established. *Alias:* `CO`
- `DROPS` Connections dropped. *Alias:* `DR`
- `CDROPS` Embryonic connections dropped. *Alias:* `CD`
- `CLOSED` Connections closed (incl. drops). *Alias:* `CL`
- `SEGSTIMED` Segments where we tried to get rtt. *Alias:* `SE`
- `RTTUPDATE` Times we succeed rtt. *Alias:* `RTT`
- `DELACK` Delayed acknowledgments sent. *Alias:* `DE`
- `TIMEODROP` Connection dropped in rxmt timeout. *Alias:* `T`
- `REXMTT` Retransmit timeouts. *Alias:* `RE`
- `PERSIST` Persist timeouts. *Alias:* `P`
- `KATIMEO` Keepalive timeouts. *Alias:* `KAT`
- `KAPROBE` Keepalive probes sent. *Alias:* `KAP`
- `KADROPS` Connections dropped in keepalive. *Alias:* `KAD`
- `STOTAL` Total packets sent. *Alias:* `ST`
- `SPACK` Data packets sent. *Alias:* `SP`
- `SBYTE` Data bytes sent. *Alias:* `SB`
- `SREPACK` Data packets retransmitted. *Alias:* `SREP`
- `SREBYTE` Data bytes retransmitted. *Alias:* `SREB`
- `SACKS` Ack-only packets sent. *Alias:* `SA`
- `SWPROBE` Window probes sent. *Alias:* `SWP`
- `SURGENT` Packets sent with URG only. *Alias:* `SU`
- `SWUPDATE` Window update-only packets sent. *Alias:* `SWU`
- `SCTRL` Control `(SYN|FIN|RST)` packets sent. *Alias:* `SC`
- `RTOTAL` Total packets received. *Alias:* `RTO`
- `RPACK` Packets received in sequence. *Alias:* `RPA`
- `RBYTE` Bytes received in sequence. *Alias:* `RB`

`RCHKSUM` Packets received with checksum errors. *Alias:* `RC`

`ROFFSET` Packets received with bad offset. *Alias:* `ROF`

`RPSHORT` Packets received too short. *Alias:* `RPS`

`RDUPPACK` Duplicate-only packets received. *Alias:* `RDUPP`

`RDUPBYTE` Duplicate-only bytes received. *Alias:* `RDUPB`

`RPDUPDATA` Packets with some duplicate data. *Alias:* `RPDUPD`

`RPDUPBYTE` Dup. bytes in part-dup. packets. *Alias:* `RPDUPB`

`ROOPACK` Out-of-order packets received. *Alias:* `ROOP`

`ROOBYTE` Out-of-order bytes received. *Alias:* `ROOB`

`RPLATE` Packets with data after window. *Alias:* `RPL`

`RBLATE` Bytes received after window. *Alias:* `RBL`

`RAFTER` Packets received after close. *Alias:* `RAF`

`RWPROBE` Received window probe packets. *Alias:* `RWP`

`RDUPACK` Received duplicate acknowledgments. *Alias:* `RDUPA`

`RACKTOOM` Received acknowledgments for unsent data. *Alias:* `RACKT`

`RACKPACK` Received acknowledgment packets. *Alias:* `RACKP`

`RACKBYTE` Bytes acknowledged by received acknowledgments. *Alias:* `RACKB`

`RWUPDATE` Received window update packets. *Alias:* `RWU`

`UDP` Variables related to User Datagram Protocol. *Alias:* `U`

`ITOTAL` Total input packets. *Alias:* `I`

`HEADSHORT` Packet shorter than header. *Alias:* `H`

`CHKSUM` Checksum error. *Alias:* `C`

`LENGTH` Data length larger than packet. *Alias:* `L`

`NOPORT` No socket on port. *Alias:* `N`

`BCNOPORT` No socket on port, arrived as broadcast. *Alias:* `B`

`FULLSOC` Not delivered, input socket full. *Alias:* `F`

`MISPCB` Input packets missing pcb cache. *Alias:* `M`

`OTOTAL` Total output packets. *Alias:* `O`

`CONNECTIONS` Returns a list of all TCP and UDP connections, including server (listening) sockets. See page 27 for the output format description.

`MBUF_STAT` Memory buffer statistics. *Alias:* `MBS`

- `MBUFS` Total number of allocated memory buffers. *Alias:* `M`
- `CLUSTERS` Total number of allocated memory buffer clusters. *Alias:* `CL`
- `CLFREE` Number of memory buffer clusters free. *Alias:* `CLF`
- `MDROPS` Times failed to find space. *Alias:* `MD`
- `NWAITED` Times waited for space. *Alias:* `NW`
- `NDRAINED` Times drained protocols for space. *Alias:* `ND`
- `TOTALMEMORYUSED` Total amount of memory used for the mbufs. *Alias:* `TMU`

`MBUF_TYPE_STATS` Returns type specific statistics of mbuf allocations. The last number is the total number of mbufs allocated. *Alias:* `MBTS`

`MBUF_CONF` Memory buffer configuration. *Alias:* `MBC`

- `INITIAL` Number of mbuf chunks to allocate initially. *Alias:* `I`
- `CHUNK` Number of mbufs to allocate at a time. *Alias:* `CH`
- `CLCHUNK` Number of clusters to allocate at a time. *Alias:* `CL`
- `MAXMEM` Maximum memory to use (in kilobytes). *Alias:* `MM`
- `CLUSTERSIZE` Size of an mbuf cluster. *Alias:* `CS`

`LOG` Logging system configuration.

- `COUNT` Number of log messages to use.
- `LEN` Maximum length of a log message.

`TASKNAME` Name of AmiTCP/IP task.

`NTHBASE` Current AmiTCP/IP has nth bsdsocket.library base currently in memory. *Alias:* `NTH`

`DEBUGSANA` Boolean to switch the SANA-II device interface debugging on and off[3]. *Alias:* `DBSANA`

[3]See page 27 for description about boolean variable.

`DEBUGICMP` Boolean to switch the ICMP debugging on and off. *Alias:* `DBICMP`

`DEBUGIP` Boolean telling whether IP should log debugging information. *Alias:* `DBIP`

`GATEWAY` Boolean to switch gateway functionality on and off. *Alias:* `GTW`

`IPSENDREDIRECTS` Boolean telling whether IP should send ICMP redirect messages. *Alias:* `REDIR`

`USENAMESERVER` How to use name server. Possible values are:

- `NO` Do not use name server at all. Local database will be used instead.
- `FIRST` Query the name servers first and if that fails, use local database.
- `SECOND` First look up the local database, then, if that fails, query the name servers.

Alias: `USENS`

`USELOOPBACK` If true use the local loop device for local traffic. *Alias:* `ULO`

`TCP_SENDSPACE` Default size of the sending socket buffer (TCP). *Alias:* `TCPSND`

`TCP_RECVSPACE` Default size of the receiving socket buffer (TCP). *Alias:* `TCPRCV`

`CONSOLENAME` Filename for the log console. *Alias:* `CON`

`LOGFILENAME` Filename for the log file. *Alias:* `LOGF`

Boolean variables

The variables documented as boolean accept their values in various formats. Boolean *false* may be given as `NO`[4], `FALSE`, `OFF` or `0`. `YES`, `TRUE`, `ON` and `1` are considered as boolean *true*.

The first alternatives are used on output.

[4] Upper case keywords are used here for clarity only. Lower case (or mixed case) keywords are accepted as well.

CONNECTIONS output format

`CONNECTIONS` query returns its result in a format which has space-separated fields, connection after connection (not sorted). Format is as follows:

Field length	Value type	Description
1	char 't' or 'u'	t for TCP connection, u for UDP
4	4-char hex	Receive queue length
4	4-char hex	Send queue length
8	8-char hex	Local address
4	4-char hex	Local port number
8	8-char hex	Foreign address
4	4-char hex	Foreign port number
1	1-char hex	State of connection

The hexadecimal values are zero padded from left to their full length. Last item presents the state of the TCP finite state machine. Possible values for it are:

Value	State of TCP FSM
0	Closed
1	Listening for connection
2	Active, has sent `SYN`
3	Has send and received `SYN`
4	Established
5	Received `FIN`, waiting for close
6	Has been closed, sent `FIN`
7	Closed exchanged `FIN`; awaiting `FIN ACK`
8	Had `FIN` and close; awaiting `FIN ACK`
9	Has been closed, `FIN` is acknowledged
A	Is in `2*msl` quiet wait after close

ICMPHIST output format

`ICMPHIST` query returns the ICMP history table entries on one line separated by spaces. Outhistory is returned first beginning from the entry 0. Both tables contains 19 entries for different ICMP messages. Note that all message types are not in use. Message types are as follows:

0	`echo` reply
1	*not used*
2	*not used*
3	Destination unreachable
4	Packet lost, slow down
5	Shorter route
6	*not used*
7	*not used*
8	Echo service
9	*not used*
10	*not used*
11	Time exceeded
12	IP header bad
13	Timestamp request
14	Timestamp reply
15	Information request
16	Information reply
17	Address mask request
18	Address mask reply

Appendix A

Autodocs for Network Applications and Utilities

The AutoDoc file **netutil.doc** contains on-line manual pages for the network utilities and applications.

Table of Contents

A.1 Network Utilities

A.1.1 arp

```
NAME
     Arp - address resolution display and control

SYNOPSIS
     arp hostname
     arp -a netname
     arp -d hostname
     arp -s hostname address [temp] [pub]
     arp -f filename

DESCRIPTION
     Arp displays and modifies the Internet to hardware address
     translation tables used by the Address Resolution Protocol.
     The hardware address is a hexadecimal string with each octet
     separated by a colon, for instance 0:12:ff:a. The length of
     the address must be same as the address of the specified
     interface.

OPTIONS
      none  If no options are specified (first form above), arp
           displays the current ARP entry for hostname.  The
           hostname must either appear in the hostname database
           (SEE hosts), or be a DARPA Internet address expressed in
           Internet standard "dot notation".

     -a    Display all current ARP entries by reading the address
           mapping table of the specified interface.

     -d   If an ARP entry exists for the host called hostname, delete
           it. This requires super-user privileges.

     -s    Create an ARP entry for the host called hostname with the
           hardware station address address. The hardware station address
           is given as six hexadecimal bytes separated by colons.  If an
           ARP entry already exists for hostname, the existing entry is
           updated with the new information.  The entry is permanent
           unless the word temp is given in the command. If the word pub
```

```
        is specified, the entry is published, which means that this
        system will act as an ARP server responding to requests for
        hostname even though the host address is not its own.

   -f   Read file filename and set multiple entries in the ARP tables.
        Entries in the file should be of the form:

             hostname address [temp] [pub]

        Argument meanings are the same as for the -s option.

AUTHOR
     Arp was developed by the University of California, Berkeley, for the
     BSD Unix system.

SEE ALSO
     ifconfig, netif.protocols/arp, "net/if_arp.h"
```

A.1.2 ifconfig

```
NAME
     ifconfig - configure network interface parameters

SYNOPSIS
     ifconfig interface address_family [address [dest_address]] [params]
     ifconfig interface [address_family]

DESCRIPTION
     ifconfig is used to assign an address to a network interface and/or
     configure network interface parameters.  ifconfig must be used at
     boot time to define the network address of each interface present on
     a machine.  It can also be used at other times to redefine an
     interface's address or other operating parameters.

PARAMETERS
     interface    A string of the unit name. The device name (e.g.
                   'a2065.device') concatenated with a slash ('/') and the
                   unit number ('11'), for example 'a2065.device/11' is a
                   legal unit name.

     address_family
                  Name of protocol on which naming scheme is based.  An
                   interface can receive transmissions in differing
                   protocols, each of which may require separate naming
                   schemes.  Therefore, it is necessary to specify the
                   address_family, which may affect interpretation of the
                   remaining parameters on the command line.  The only
                   address family currently supported is inet (DARPA-
                   Internet family).

     address      Either a host name present in the host name database,
                   (SEE hosts), or a DARPA Internet address
                   expressed in Internet standard "dot notation".  The
                   host number can be omitted on 10-Mbyte/second Ethernet
                   interfaces (which use the hardware physical address),
                   and on interfaces other than the first.

     dest_address Address of destination system.  Consists of either a
                   host name present in the host name database, hosts(4),
                   or a DARPA Internet address expressed in Internet
```

```
                 standard "dot notation".

 SWITCHES
      The following operating parameters can be specified:

       up            Mark an interface "up". Enables interface after an
                     "ifconfig down."  Occurs automatically when setting the
                     address on an interface.  Setting this flag has no
                     effect if the hardware is "down".

       down          Mark an interface "down".  When an interface is
                     marked "down", the system will not attempt to
                     transmit messages through that interface. If
                     possible, the interface will be reset to disable
                     reception as well.  This action does not
                     automatically disable routes using the interface.

      arp            Enable the use of Address Resolution Protocol in
                     mapping between network level addresses and link-level
                     addresses (default).

      -arp           Disable the use of Address Resolution Protocol.

       metric n      Set the routing metric of the interface to n,
                     default 0.  The routing metric is used by the routing
                     protocol (see gated(1m)).  Higher metrics have the
                     effect of making a route less favorable; metrics are
                     counted as additional hops to the destination network
                     or host.

       debug         Enable driver-dependent debugging code. This usually
                     turns on extra console error logging.

      -debug         Disable driver-dependent debugging code.

       netmask mask (Inet only) Specify how much of the address to reserve
                     for subdividing networks into sub-networks.  mask
                     includes the network part of the local address, and the
                     subnet part which is taken from the host field of the
                     address.  mask can be specified as a single hexadecimal
                     number with a leading 0x, with a dot-notation Internet
```

```
                address, or with a pseudo- network name listed in the
                network table networks(4).  mask contains 1's for each
                bit position in the 32-bit address that are to be used
                for the network and subnet parts, and 0's for the host
                part.  mask should contain at least the standard
                network portion, and the subnet field should be
                contiguous with the network portion.

     broadcast   (Inet only) Specify the address that represents
                broadcasts to the network.  The default broadcast
                address is the address with a host part of all 1's.

     The command:

          ifconfig interface/unit

     with no optional command arguments supplied displays the current
     configuration for interface.  If address_family is specified,
     ifconfig reports only the details specific to that address family.

DIAGNOSTICS

     Messages indicating that the specified interface does not exist, the
     requested address is unknown, or the user is not privileged and
     tried to alter an interface's configuration.

EXAMPLES

     ifconfig lo/0 127.0.0.1

             This command marks internal loopback device "UP", and
             attach an inet address 127.0.0.1 to it.

     ifconfig cslip.device/0 inet 193.102.4.144 193.102.4.129

             This command starts the CSLIP driver, attach an
             address 193.102.4.144 (our internet address) and a
             destination address 193.102.4.129 (the internet
             address of the host you are connecting) to it.

     ifconfig devs:network/a2065.device/0 inet 193.124.100.64 +
```

```
     netmask 255.255.255.192 -arp

             This command loads an ethernet driver for the
             Commodore A2065 Ethernet adapter unit 0, marks it
             "up", disables ARP protocol for it, attaches an inet
             address 193.124.100.65 to it and sets its netmask to
             255.255.255.192.  A bitwise logical and of netmask and
             address for the interface forms a subnet address, in
             this case 193.124.100.64.  All packets aimed to hosts
             with same subnet address (that is hosts 193.124.100.64
             - 193.124.100.127) are routed to this interface.

SEE ALSO
     netstat, hosts, arp, "net/if.h", "net/sana2tags.h"
```

A.1.3 inetd

```
NAME
       inetd - internet ``super-server''

TEMPLATE
      inetd DEBUG/S CONFIGFILE

DESCRIPTION

      Inetd should be run when the AmiTCP/IP protocol stack is started.
      Inetd listens for connections on certain internet sockets.  When a
      connection is found on one of its sockets, it decides what service the
      socket corresponds to, and invokes a program to service the request.
      After the program is finished, it continues to listen on the socket
      (except in some cases which will be described below).  Essentially,
      inetd allows running one daemon to invoke several others, reducing
      load on the system.

PARAMETERS
      DEBUG      Turns on debugging.

      CONFIGFILE Specifies the configuration file name.

CONFIGURATION

      Upon execution, inetd reads its configuration information from a
      configuration file which, by default, is AmiTCP:db/inetd.conf. There
      must be an entry for each field of the configuration file, with
      entries for each field separated by a tab or a space.  Comments are
      denoted by a ``#'' at the beginning of a line or ``;'' anywhere in the
      line. There must be an entry for each field. The fields of the
      configuration file are as follows:

            service name
            socket type
            protocol
            wait/nowait
            user
            server program
            server program name
            server program arguments
```

```
The service-name entry is the name of a valid service in the
netdatabase.  For ``internal'' services (discussed below), the service
name must be the official name of the service.

The socket-type should be one of ``stream'', ``dgram'', ``raw'',
``rdm'', or ``seqpacket'', depending on whether the socket is a
stream, datagram, raw, reliably delivered message, or sequenced packet
socket. Current system supports only stream, datagram and raw
protocols.

The protocol must be a valid protocol as given in netdatabase.
Examples might be ``tcp'' or ``udp''.

The wait/nowait entry is useful for datagram sockets only (other
sockets should have a ``nowait'' entry in this space).  If a datagram
server connects to its peer, freeing the socket so inetd can received
further messages on the socket, it is said to be a ``multi-threaded''
server, and should use the ``nowait'' entry.  For datagram servers
which process all incoming datagrams on a socket and eventually time
out, the server is said to be ``single-threaded'' and should use a
``wait'' entry.  Comsat and talkd are both examples of the latter type
of datagram server.

The user entry should contain the user name of the user as whom the
server should run. This field is for Unix and future compability
only.

The server-program entry should contain the pathname of the program
which is to be executed by inetd when a request is found on its
socket.  If the server program is resident, the path name should be
suppressed. If inetd provides this service internally, this entry
should be ``internal''.

The server-program-name is CLI command name for the server process. It
is shown in the printout of ``status'' command. (Task name of the
server process is the service and the peer address, e.g. ``echo
[192.233.15.19]''.) This and argument entry are optional.

The server program arguments should be just as arguments normally are.
```

Inetd provides several ``trivial'' services internally by use of routines within itself. These services are ``echo'', ``discard'', ``chargen'' (character generator), ``daytime'' (human readable time), and ``time'' (machine readable time, in the form of the number of seconds since mid night, January 1, 1900). All of these services are TCP and UDP based. For details of these services, consult the appropriate RFC from the Network Information Center.

Inetd rereads its configuration file when it receives the CTRL-F signal. Services may be added, deleted or modified when the configuration file is reread.

HISTORY

The inetd command appeared in 4.3BSD system.

SEE ALSO

A.1.4 letnet

```
NAME
     Letnet - a simple TCP connection tool

SYNOPSIS
     letnet HOSTNAME/A,PORT/A

DESCRIPTION
     Letnet connects to the specified TCP port at the specified host. The
     data read from standard input is sent to the host and data received
     from the connection is written to the standard output. Letnet
     terminates upon shutdown of the socket or receiving SIGBREAKF_CTRL_C
     signal.

ARGUMENTS
      HOSTNAME/A
             If there is no name service available, hostname may be given
             in the Internet dot notation.

      PORT/A
             The port identifier is searched from the standard services
             (SEE ALSO netdb/services) database. A nonstandard
             service port may be specified in the numeric form, numbers
             between 1---65535 are acceptable.

AUTHOR
     Pekka Pessi, the AmiTCP/IP Group, Helsinki University of Technology

SEE ALSO
     netdb/services, netdb/hosts
```

A.1.5 offline

```
NAME
     Offline - put a SANA-II device offline

TEMPLATE
     Offline DEV=DEVICE devicename[/unit] [UNIT unit]

DESCRIPTION
     Offline sends the S2_OFFLINE command to the given SANA-II device
     driver and unit. This command is normally used to disconnect SANA-II
     device driver from the network adapter hardware. Device driver does
     not accept any more read or write requests.

EXAMPLES
     This command puts the SLIP offline, which frees then the serial port
     to your use.

     OFFLINE slip.device/1

SEE ALSO
     Online, sana2.device/S2_OFFLINE
```

A.1.6 online

```
NAME
     Online - put a SANA-II device online

TEMPLATE
     Online DEV=DEVICE devicename[/unit] [UNIT unit]

DESCRIPTION
     Online sends the S2_ONLINE command to the given SANA-II device driver
     and unit. The device driver restarts the network adapter hardware and
     accepts read and write request again.

EXAMPLES

     This command puts the Commodore Ethernet driver online.

     Online a2065.device/0

SEE ALSO
     Offline, sana2.device/S2_ONLINE
```

A.1.7 ping

```
NAME
    ping - send ICMP ECHO_REQUEST packets to network hosts

SYNOPSIS
    ping [-dfnqrvR] [-c count] [-i wait] [-l preload] [-p pattern]
         [-s packetsize]

DESCRIPTION
    Ping uses the ICMP protocol's mandatory ECHO_REQUEST datagram to
    elicit an ICMP ECHO_RESPONSE from a host or gateway.  ECHO_REQUEST
    datagrams (``pings'') have an IP and ICMP header, followed by a
    ``struct timeval'' and then an arbitrary number of ``pad'' bytes
    used to fill out the packet.  The options are as follows: Other
    options are:

    -c count
            Stop after sending (and receiving) count ECHO_RESPONSE
            packets.

    -d      Set the SO_DEBUG option on the socket being used.

    -f      Flood ping.  Outputs packets as fast as they come back or
            one hundred times per second, whichever is more.  For every
            ECHO_REQUEST sent a period ``.'' is printed, while for ever
            ECHO_REPLY received a backspace is printed.  This provides a
            rapid display of how many packets are being dropped.  Only
            the super-user may use this option.  This can be very hard
            on a network and should be used with caution.

    -i wait
            Wait wait seconds between sending each packet. The default
            is to wait for one second between each packet.  This option
            is incompatible with the -f option.

    -l preload
            If preload is specified, ping sends that many packets as
            fast as possible before falling into its normal mode of
            behavior.

    -n      Numeric output only.  No attempt will be made to lookup
```

```
        symbolic names for host addresses.

-p pattern
        You may specify up to 16 ``pad'' bytes to fill out the
        packet you send.  This is useful for diagnosing
        data-dependent problems in a network.  For example, ``-p
        ff'' will cause the sent packet to be filled with all ones.

-q      Quiet output.  Nothing is displayed except the summary lines
        at startup time and when finished.

-R      Record route.  Includes the RECORD_ROUTE option in the
        ECHO_REQUEST packet and displays the route buffer on
        returned packets.  Note that the IP header is only large
        enough for nine such routes.  Many hosts ignore or discard
        this option.

-r      Bypass the normal routing tables and send directly to a host
        on an attached network.  If the host is not on a
        directly-attached network, an error is returned.  This
        option can be used to ping a local host through an interface
        that has no route through it.

-s packetsize
        Specifies the number of data bytes to be sent.  The default
        is 56, which translates into 64 ICMP data bytes when
        combined with the 8 bytes of ICMP header data.

-v      Verbose output.  ICMP packets other than ECHO_RESPONSE that
        are received are listed.

When using ping for fault isolation, it should first be run on the
local host, to verify that the local network interface is up and
running.  Then, hosts and gateways further and further away should
be ``pinged''.  Round-trip times and packet loss statistics are
computed.  If duplicate packets are received, they are not included
in the packet loss calculation, although the round trip time of
these packets is used in calculating the minimum/average/maximum
round-trip time numbers.  When the specified number of packets have
been sent (and received) or if the program is terminated with a
SIGINT, a brief summary is displayed.
```

```
     This program is intended for use in network testing, measurement and
     management.  Because of the load it can impose on the network, it is
     unwise to use ping during normal operations or from automated
     scripts.

ICMP PACKET DETAILS
     An IP header without options is 20 bytes.  An ICMP ECHO_REQUEST
     packet contains an additional 8 bytes worth of ICMP header followed
     by an arbitrary amount of data.  When a packetsize is given, this
     indicated the size of this extra piece of data (the default is 56).
     Thus the amount of data received inside of an IP packet of type ICMP
     ECHO_REPLY will always be 8 bytes more than the requested data space
     (the ICMP header).

     If the data space is at least eight bytes large, ping uses the first
     eight bytes of this space to include a timestamp which it uses in
     the computation of round trip times.  If less than eight bytes of
     pad are specified, no round trip times are given.

DUPLICATE AND DAMAGED PACKETS
     Ping will report duplicate and damaged packets.  Duplicate packets
     should never occur, and seem to be caused by inappropriate
     link-level retransmissions.  Duplicates may occur in many situations
     and are rarely (if ever) a good sign, although the presence of low
     levels of duplicates may not always be cause for alarm.

     Damaged packets are obviously serious cause for alarm and often
     indicate broken hardware somewhere in the ping packet's path (in the
     network or in the hosts).

TRYING DIFFERENT DATA PATTERNS
     The (inter)network layer should never treat packets differently
     depending on the data contained in the data portion.  Unfortunately,
     data-dependent problems have been known to sneak into networks and
     remain undetected for long periods of time.  In many cases the
     particular pattern that will have problems is something that doesn't
     have sufficient ``transitions'', such as all ones or all zeros, or a
     pattern right at the edge, such as almost all zeros.  It isn't
     necessarily enough to specify a data pattern of all zeros (for
     example) on the command line because the pattern that is of interest
```

is at the data link level, and the relationship between what you type and what the controllers transmit can be complicated.

This means that if you have a data-dependent problem you will probably have to do a lot of testing to find it. If you are lucky, you may manage to find a file that either can't be sent across your network or that takes much longer to transfer than other similar length files. You can then examine this file for repeated patterns that you can test using the -p option of ping.

TTL DETAILS

The TTL value of an IP packet represents the maximum number of IP routers that the packet can go through before being thrown away. In current practice you can expect each router in the Internet to decrement the TTL field by exactly one.

The TCP/IP specification states that the TTL field for TCP packets should be set to 60, but many systems use smaller values (4.3 BSD uses 30, 4.2 used 15). The AmiTCP/IP uses normally TTL value 30.

The maximum possible value of this field is 255, and most systems set the TTL field of ICMP ECHO_REQUEST packets to 255. This is why you will find you can ''ping'' some hosts, but not reach them with telnet or ftp.

In normal operation ping prints the ttl value from the packet it receives. When a remote system receives a ping packet, it can do one of three things with the TTL field in its response:

Not change it; this is what Berkeley Unix systems did before the 4.3BSD-Tahoe release. In this case the TTL value in the received packet will be 255 minus the number of routers in the round-trip path.

Set it to 255; this is what AmiTCP/IP and current Berkeley Unix systems do. In this case the TTL value in the received packet will be 255 minus the number of routers in the path from the remote system to the pinging host.

Set it to some other value. Some machines use the same value for ICMP packets that they use for TCP packets, for example

```
        either 30 or 60.  Others may use completely wild values.

BUGS
    Many Hosts and Gateways ignore the RECORD_ROUTE option.

    The maximum IP header length is too small for options like
    RECORD_ROUTE to be completely useful.  There's not much that that
    can be done about this, however.

    Flood pinging is not recommended in general, and flood pinging the
    broadcast address should only be done under very controlled
    conditions.

SEE ALSO
    netstat,  ifconfig

AUTHOR
    The ping command originally appeared in 4.3BSD.
```

A.1.8 route

```
NAME
     route - manually manipulate the routing tables

SYNOPSIS
     route [-n] [-q] [-v] command [modifiers] destination gateway

DESCRIPTION
     Route is a program used to manually manipulate the network routing
     tables.

     Options supported by route:

     -n      Prevent attempts to print host and network names
             symbolically when reporting actions.

     -v      (verbose) Print additional details.

     -q      Suppress all output.

     Commands accepted by route:

     add         Add a route.
     delete      Delete a specific route.

     The destination is the destination host or network, gateway is the
     next-hop gateway to which packets should be addressed. Routes to a
     particular host are distinguished from those to a network by
     interpreting the Internet address associated with destination. The
     optional modifiers -net and -host force the destination to be
     interpreted as a network or a host, respectively.  Otherwise, if the
     destination has a ``local address part'' of INADDR_ANY, or if the
     destination is the symbolic name of a network, then the route is
     assumed to be to a network; otherwise, it is presumed to be a route
     to a host.

     For example, 128.32 is interpreted as -host 128.0.0.32; 128.32.130
     is interpreted as -host 128.32.0.130; -net 128.32 is interpreted as
     128.32.0.0; and -net 128.32.130 is interpreted as 128.32.130.0.

     To add a default route, give the destination as 'default'.
```

```
     If the route is via an interface rather than via a gateway, the
     -interface modifier should be specified; the gateway given is the
     address of this host on the common network, indicating the interface
     to be used for transmission.

     The optional -netmask qualifier is used to specify the netmask of
     the interface. One specifies an additional ensuing address parameter
     (to be interpreted as a network mask).  The implicit network mask
     generated can be overridden by making sure this option follows the
     destination parameter.

     All symbolic names specified for a destination or gateway are looked
     up first as a host name using gethostbyname(). If this lookup fails,
     getnetbyname() is then used to interpret the name as that of a
     network.

DIAGNOSTICS
     add [host | network ] %s: gateway %s flags %x
             The specified route is being added to the tables. The values
             printed are from the routing table entry supplied in the
             IoctlSocket() call. If the gateway address used was not the
             primary address of the gateway (the first one returned by
             gethostbyname()), the gateway address is printed numerically
             as well as symbolically.

     delete [ host | network ] %s: gateway %s flags %x
             As above, but when deleting an entry.

     Network is unreachable
             An attempt to add a route failed because the gateway listed
             was not on a directly-connected network.  The next-hop
             gateway must be given.

     not in table
             A delete operation was attempted for an entry which wasn't
             present in the tables.

     routing table overflow
             An add operation was attempted, but the system was low on
             resources and was unable to allocate memory to create the
```

```
          new entry.

SEE ALSO
     ifconfig, protocols/routing

HISTORY
     The route command appeared in 4.2BSD.
```

Glossary

API *Application Program Interface.* Standard function calls, messages and devices used in application level programs.

Arcnet Yet another network type, transfer rate 5Mbits/sec.

ARexx ARexx is a specific implementation of the Rexx language for the Amiga. Rexx itself is a script language with superb power/simplicity ratio, originally developed for some IBM mainframe systems.

ARP *Address Resolution Protocol.* A communication protocol used to map protocol address to network address dynamically. For example, ARP is used to map DARPA Internet addresses into Ethernet addresses.

AutoDoc Document generated automagically from comments included in program source code. Mainly used to describe shared library functions in AmigaOS. See [RKM Inc & ADoc 1992].

BSD *Berkeley Software Distribution.* Family of UNIX versions for the DEC (VAX) and PDP-11 developed by Bill Joy and others at Berzerkeley starting around 1980, incorporating paged virtual memory, TCP/IP networking enhancements, and many other features. The BSD versions (4.1, 4.2, and 4.3) and the commercial versions derived from them (SunOS, ULTRIX, and Mt. Xinu) held the technical lead in the UNIX world until AT&T's successful standardization efforts after about 1986, and are still widely popular.

BSDSS *BSD Single Server* is operating system server in Mach operating system where all UNIX services are in single binary.

(D)ARPA Internet The collection of networks using internet protocols. (D)ARPA Internet originated from a research project sponsored by the (Defense) Advanced Research Project Agency.

Daemon 'A deified being', program that lives forever on system, or reincarnates every time it is needed. Performs its requested task promptly and goes back to sleep.

Data link layer Protocol layer providing data transfer between machines in same physical network.

DIS First stage in ISO's standardization. (Discussion)

Ethernet 10Mbit/sec physical network interface. Developed by Xerox in Palo Alto late 70s. In 1982 published with Digital and Intel as ESPEC1. ESPEC2 was approved as basic specification for LANs which employed the carrier sense multiple access with collision detection by IEEE 802.3 committee. Heavily derived version was later published by ISO as DIS 8802/3. Original has only version for 50 Ω coaxial cable, 10BASE5 "Thick Ether", but nowadays has also for thinner coaxial cable, 10BASE2 "Thin Ether", twisted pair, 10BASET and fibre, 10BASEF. There are also broadband, 10BROAD36, and slower, 1Mbit/s 1BASE5, versions.

EXEC AmigaOS *EXECutive*. Provides the basic kernel functions, such as task scheduling, memory management, concurrency control, message ports, public library lists etc.

Frame Data unit transferred between data link layer protocol entities.

ICMP *Internet Message Control Protocol.* A host-to-host communication protocol used in the DARPA Internet for reporting errors and controlling the operation of IP.

IEEE the Institute of Electrical and Electronics Engineers

Inetd the Internet super server. A server that listens to the network and launches other server processes when needed.

IO request An Amiga standard message format to discuss with devices. IO requests are sent by Exec function `BeginIO()` to the device driver. Completed IO requests are sent back to the reply port specified in IO request.

IP *Internet Protocol.* The network-layer communication protocol used in the DARPA Internet. IP is responsible for host-to-host addressing and routing, packet forwarding, and packet fragmentation and reassembly.

ISO International Organization for Standardization

Jargon A formal technical vocabulary of various subjects. Also hackish slang or mumbo jumbo is referred as jargon [Raymond 1992].

Library base In AmigaOS shared library is accessed via a library base pointer. On the negative side (relative to the base pointer) there is jump table to the library functions and on the positive side – library data the functions can use.

Log It is usually useful to record information. If something has gone wrong, one may found information to remove cause of failure. A log is usually a file, where information is written.

Mbuf *Memory BUFfer.* Data storage unit inside BSD networking implementation.

Message Data unit transferred between applications using network services.

MTU *Maximum Transfer Unit.* The amount of data that underlying network can transfer in one block.

NET/2 Latest (by now) BSD Unix release.

Network File System A filesystem share protocol which enables other computers use disks on other host over network. Usually uses UDP protocol for transfering files, but also TCP-based version exists. Developed by Sun Microsystems Inc., and developed to "Industrial Standard" in *IX environment. Described in detail "Networking on the Sun Workstation"

Octet A unit of eigth bits. Used in communication to overcome problem that byte is not allways eigth bits (althoug this is nowadays very rare situation.)

Packet Data unit transferred between network layer protocol entities.

`panic()` Function executed after lethal failure in Unix kernel.

PPP *Point-to-Point Protocol* which allows multiple protocols to be transferred on the same line. It is also more flexible than SLIP.

Protocol stack A network protocol stack is a layer of software that network applications use to address particular processes on remote machines. The AMITCP/IP is such a protocol stack using TCP/IP protocols.

RARP *Reverse Address Resolution Protocol* maps network addresses to protocol addresses dynamically.

RFC *Request For Comments.* Freely available standards for networking. They are mostly available online from `nic.ddn.mil` with **FTP** or **Kermit**.

RKM *Rom Kernel reference Manuals* are the most authoritative AmigaOS references, see [RKM Libraries 1992] for example.

SANA-II A standard for an Amiga software interface between networking hardware and network protocol stacks (or for software tools such as network monitors).

SLIP *Serial Line Internet Protocol.* Data encapsulation protocol used to transmit IP packets over point-to-point serial lines.

Socket An abstraction for endpoint of communication.

TLA *Three Letter Acronym*, a mnemonic and mystic abbreviation which is coined to confuse acolytes.

TCP *Transmission Control Protocol.* A connection-oriented transport protocol used in the DARPA Internet. TCP provides for the reliable transfer of data, as well as the out-of-band indication of urgent data.

TCP/IP The whole protocol stack, which implements at least the Internet protocols required is often expressed as the two most commonly used, TCP/IP

UDP *User Datagram Protocol.* A simple unreliable datagram protocol used in the DARPA Internet. UDP provides only peer-to-peer addressing and optional data checksum.

Wire type is the physical layer protocol type. Different wire types are for instance Arcnet, Ethernet, IEEE 802.3, PPP and SLIP.

Bibliography

[Comer 1988] Comer, D. E., *Internetworking With TCP/IP Vol I: Principles, Protocols and Architecture,* Prentice–Hall International, 382 p.

[Comer and Stevens 1991] Comer, D. E. and Stevens, D. L., *Internetworking With TCP/IP Vol II: Design, Implementation, and Internals,* Prentice–Hall International, 524 p.

[Finlayson et al 1984] Finlayson, R., Mann, T., Mogul, J. and Theimer, M., "A Reverse Address Resolution Protocol," RFC 903, 4 p.

[Holloway 1991] Holloway, T., "Object Oriented Amiga EXEC," *BYTE*, vol. 16, issue 1, pp. 329–334, January 1991.

[Jacobson 1990] Jacobson, V., "Compressing TCP/IP Headers for Low-speed Serial Links," RFC 1144, 43 p.

[Leffler et al 1989] Leffler, S. J., McKusick, M. K., Karels, M. J. and Quarterman, J. S., *The Design and Implementation of the 4.3BSD UNIX Operating System,* Addison–Wesley 471 p.

[Leffler et al 1991a] Leffler, S. J., Fabry, R. S., Joy, W. N., Lapsley, P., Miller, S., Torek, C., "An Advanced 4.3BSD Interprocess Communication Tutorial" UNIX Programmer's Supplementary Documents (PS1), net/2 Berkeley Software Distribution, Computer Systems Research Group, Univ. of California, Berkeley, 53 p.

[Leffler et al 1991b] Leffler, S. J., Joy, W. N., Fabry, R. S. and Karels, M. J., "Networking Implementation Notes, 4.3BSD Edition," UNIX System Manager's Manual (SMM), net/2 Berkeley Software Distribution, Computer Systems Research Group, Computer Science Division, Univ. of California, Berkeley, 26 p.

[Plummer 1982] Plummer, D. C., "An Ethernet Address Resolution Protocol," RFC 826, 10 p.

[Postel 1980] Postel, J., "User Datagram Protocol," RFC 768, 3 p.

[Postel 1981a] Postel, J. ed., "Internet Protocol," RFC 791, 45 p.

[Postel 1981b] Postel, J., "Internet Control Message Protocol," RFC 792, 21 p.

[Postel 1981c] Postel, J. ed., "Transmission Control Protocol," RFC 793, 85 p.

[Raymond 1992] Raymond, Eric ed., "The Jargon File 2.9.9", the on-line hacker Jargon File, version 2.9.9, 01 APR 1992.

[Reynolds 1990] Reynolds, Joyce K., "Assigned Numbers", RFC 1060, 86 p.

[RKM Libs & Devs 1989] Commodore–Amiga Inc., *Amiga ROM Kernel Reference Manual: Libraries and Devices,* Addison Wesley, 956 p.

[RKM Libraries 1992] Commodore–Amiga Inc., *Amiga ROM Kernel Reference Manual: Libraries, 3rd ed.,* Addison Wesley, 967 p.

[RKM Inc & ADoc 1992] Commodore–Amiga Inc., *Amiga ROM Kernel Reference Manual: Includes & AutoDocs, 3rd ed.,* Addison Wesley, 471 p.

[SANA-II 1992] Amiga Networking Group, "SANA-II Network Device Driver Specification - Rev 1.0 23-Apr-92," published at *Fred Fish*-collection disk#673.

[SANA-II 1992 add] Amiga Networking Group, "Addenda to SANA-II Network Device Driver Specification," published at SANA-II Developer Support Package, May 21, 1992.

[Stevens 1990] Stevens, W., R., *UNIX network programming,* Prentice Hall, 772 p.

[Winsock 1992] Hall, M., Towfiq, M., Arnold, G., Trendwell, D., Sanders, H., "Windows Sockets An Open Interface for Network Programming under Microsoft Windows Version 1.0 Rev.A" 11 June, 1992, 124 p.

The RFC documents are stored online on `nic.ddn.mil`, from where they can be downloaded with anonymous FTP. The file **/rfc/rfc-index.txt** contains index to all published RFCs.

The BSD documents are carried by many FTP sites, for example `nic.funet.fi`.