The Fuzzball[1] [2]

David L. Mills
Electrical Engineering Department
University of Delaware

Abstract

The Fuzzball is an operating system and applications library designed for the PDP11 family of computers. It was intended as a development platform and research pipewrench for the DARPA/NSF Internet, but has occasionally escaped to earn revenue in commercial service. It was designed, implemented and evolved over a seventeen-year era spanning the development of the ARPANET and TCP/IP protocol suites and can today be found at Internet outposts from Hawaii to Italy standing watch for adventurous applications and enduring experiments. This paper describes the Fuzzball and its applications, including a description of its novel congestion avoidance/control and timekeeping mechanisms.

Keywords: protocol testing, network testing, performance evaluation, Internet architecture, TCP/IP protocols, congestion control, internetwork time synchronization.

1. Introduction

The Fuzzball is a software package consisting of a fast, compact operating system, support for the DARPA/NSF Internet architecture [8] and an array of application programs for network protocol development, testing and evaluation. It usually runs on a LSI-11 personal workstation, to which it also lends its name, and functions as a multi-purpose packet switch, gateway and service host. It supports the complete TCP/IP protocol suite, including network, transport and applications-level protocols for virtual-terminal, file-transfer and mail systems with text, voice and image capabilities. It includes a comprehensive user interface based on a multiple-user, virtual-machine emulator for the DEC RT-11 operating system, so that RT-11 program-development utilities and user programs can be run along with network-application programs in an internetworking environment.

It is not clear how or exactly when the Fuzzball came to be called that. Its ancestor was born at the University of Edinburgh in 1971 and evolved as the DCN operating system for a distributed network of PDP11 virtual machines at the University of Maryland in 1975 [MIL75]. Among the lessons learned is the fact that context switching in a virtualized PDP11 architecture is painfully slow, so the DCN system remained a demonstration curio.

By 1977 the DCN system was dismantled and rebuilt in a leaner, zippier design and software was developed for the TCP/IP protocol suite. The DCN routing protocol, now called Hellospeak [12], was completely redesigned to link DCN hosts and gateways together and to other networks, including ARPANET, SATNET and various LANs. At the time there were no full-featured Internet hosts other than DCN based on a microprocessor; so, presumably because nobody knew what DCN stood for, they became affectionally and widely known as Fuzzballs. The Fuzzball has since been in a state of continuous, reckless evolution to support more processor features and peripherals and to support progressively more complex protocol suites and more sophisticated resource-management techniques.

Over the years many network experiments, demonstrations and more durable applications have been mounted on the Fuzzball, a selection of which is described in subsequent sections. Several experiments with advanced-technology networks such as packet radio, packet satellite and various gateway systems have used Fuzzballs as debugging, measurement and evaluation platforms. Various organizations have used Fuzzballs as terminal concentrators, mail and word-processing hosts, network monitoring and control devices and general-purpose packet switches and gateways. Fuzzballs have seen regular service on the INTELPOST and NSFNET Backbone networks and several campus networks at university, government and industrial research laboratories.

Perhaps the most useful and interesting Fuzzball application is as an experimental tool for the development and

1. Reprinted from: Mills, D.L. The Fuzzball. *Proc. ACM SIGCOMM 88 Symposium* (Palo Alto CA, August 1988, 115-122).

2. Sponsored by: Defense Advanced Research Projects Agency contract number N00140-87-C-8901 and by National Science Foundation grant number NCR-86-12015.

evaluation of novel network algorithms and protocols. Many Fuzzballs have been deployed for that purpose at various locations in the US and Europe. In the typical case several Fuzzballs are lashed together with Ethernets and makeshift wires, then connected to the Internet via an existing LAN or gateway. Fuzzballs have even been connected via dial-up telephone circuits, amateur packet-radio channels, international undersea cables, satellite terminals and strange statistical multiplexors.

This paper begins with an overfly of the Fuzzball system architecture, including process and memory structure, interprocess communication, and network support. Included are brief descriptions of application software found useful for network testing, debugging and evaluation. The paper then summarizes selected experiments and demonstrations in which Fuzzballs have played a pivotal part and highlights the useful features and lessons learned. It describes how Fuzzballs have been used in operational systems such as the INTELPOST and NSFNET Backbone networks and concludes with a detailed description of two interesting applications, one involving congestion avoidance/control and the other the synchronization of network clocks.

2. System Architecture

The Fuzzball operating system consists of the supervisor and two kinds of processes, supervisor and user. The supervisor consists of the process scheduler and a collection of supervisor primitives (EMT instructions) used by all processes. Supervisor processes are used to support direct-access and terminal devices, as well as network interfaces and protocol multiplexors. User processes provide an emulated environment in which most system and user programs developed for the RT-11 operating system can run unchanged. Some user processes support dedicated functions such as routing daemons, input/output spoolers and generic network servers supporting connection-based (TCP) and connectionless (UDP) services.

A process consists of four virtual-memory segments: the instruction segment, data segment, descriptor segment and parameter segment. The instruction and data segments for the supervisor and supervisor processes are allocated in separate kernel spaces to provide the maximum memory possible for buffers, control blocks and tables. The instruction and data segments for each user process are allocated in private user spaces, except for the emulator code, which resides in shared segments mapped into user space at addresses usually reserved for the RT-11 resident monitor.

Descriptor segments contain the process stack and memory-management descriptors, as well as data used for process scheduling, interprocess communication and related functions. These segments are allocated in kernel space and not normally accessible to application programs. Parameter segments contain data used to establish various operational parameters, such as interrupt vector, disk volume size, device timeout and so forth, as well as various throughput and error counters, depending on process type. These segments are also allocated in kernel space, but mapped into each user process adjacent to the emulator code. This is done in order to facilitate operator monitoring and control of the various processes and their operational parameters.

Virtual windows are used to access device registers and to share buffers and control blocks between processes to avoid time-consuming copy operations. Mutually exclusive access to shared data areas is provided using a set of semaphore queues supported by supervisor primitives. Processes encountering an occupied critical section are blocked and eventually serviced in order of arrival. A fixed number of semaphores is available for specific use, such as update access to the routing data base, volume directories and so forth.

Processes communicate with one another using small (16-byte) interprocess messages in one of four formats. Message passing is supported by supervisor primitives that send a message or wait for one to arrive. Sending never blocks the process, while receiving does, unless a message is already waiting or until a specified timeout expires. Processes can send special messages called asynchronous interrupts which result from exceptional events and are especially useful for system logging and operator alarms.

The scheduler determines the next process to be run, restores the processor state from the descriptor segment and activates the process. It uses a multi-level priority service discipline with preemptive round-robin (timesliced) service at each priority level. Preemptions occur when a higher priority process is scheduled due to a message arrival, semaphore unblock, timeslice end, asynchronous event or voluntarily. Voluntary preemptions result in the process either waiting for a message, semaphore unblock or asynchronous event such as a timer interrupt.

The emulator intercepts synchronous interrupts (EMT instructions, etc.) and asynchronous interrupts and presents them within the virtual environment of the process. The user process interface presented by the emulator is a subset of that presented by the resident monitor of the RT-11 system and includes all of the programmed requests necessary to run standard RT-11 system components and user programs.

A highly evolved logical clock is an intrinsic feature of the Fuzzball [16]. The logical clock increments at 1000 Hz and is equipped with frequency and phase control mechanisms useful for synchronizing time to other systems with either the Hellospeak routing protocol or Net-

work Time Protocol mentioned later in this paper. Besides the logical clock, the system provides per-process timers and a full complement of timer and clock primitives.

2.1. Network Software

A pair of supervisor device-driver processes is dedicated to every network interface device, one for input and one for output. There are three levels to a device driver, a common one responsible for Internet processing, including routing, fragmentation and various error processing, another below it responsible for the particular local-net architecture, such as Ethernet, serial line and so forth, and the third, which amounts to a set of interrupt routines and is responsible for the operation of the particular device. Support for the Hellospeak routing protocol is provided as an optional module that operates at the Internet level.

The design is such that packets are transferred from the input device directly to a buffer and then out from the same buffer directly to the output device and without copying. This modular structure allows considerable flexibility, since the modules at the various levels can be swapped as required for the local net and device required.

The TCP and UDP transport modules are implemented as a dedicated supervisor process, which is identified by an assigned Internet address. All datagrams directed to this address are delivered to this process, which is then responsible for matching to the correct protocol and port, assembling/disassembling the data and delivering it to a user process. A host can have multiple processes of this type in order to support configurations where the host is connected to more than one network interface. Transfers to and from the user processes use a shared virtual window in which data are copied directly between kernel-space buffers and user-space buffers.

Some protocols are supported at the user-process level, including a UDP daemon for file-transfer, name-resolution and time protocols, as well as an EGP daemon. TCP daemons are used for remote spooling, both send and receive, to support standard and multi-media mail systems, as well as remote print, voice and image functions for special devices. TCP services such as the virtual-terminal (TELNET) server, file-transfer (FTP) server and several miscellaneous servers are handled by assigning a user process to the service upon arrival of a connection request and then passing an appropriate command to the application program which controls that process.

2.2. Application Software

Each user has access to one or more user processes and can switch a single terminal from one to the other in a few keystrokes. A conventional command language interpreter (shell) running in each user process is used to initiate application programs, manage debug sessions, get system help and in general run the system. A very useful feature is that one process can be used to debug another while both are running. In addition, input and output streams can be redirected and virtual environments can be altered. A virtual volume feature allows volumes, complete with directory, to be encapsulated as an ordinary file.

Among the often-used application programs supported by connection-oriented transport service (TCP) are the following:

TELNET virtual-terminal protocol client and server programs, which provide user access to all Fuzzball functions, including command, control, monitoring and process/operator intercommunication. The client program includes features to test the negotiation mechanisms peculiar to that protocol, and to record and playback session text and emulate graphics display devices.

FTP file-transfer protocol client and server programs, which provide the basic means to exchange program and data files between Fuzzballs in both ASCII and binary modes.

Server and utility programs for exchanging mail with other Internet hosts. One of these is based on the ubiquitous (text-only) SMTP mail protocol used throughout the Internet, while the other uses the experimental MPM multi-media mail protocol and supports text, image, graphics and real-time speech modes.

Utility servers for various "little" services, including TCP echo, discard, text generator and various user and system information utilities. In addition, a comprehensive Unix-compatible spooling facility can be used with text, image and speech devices.

Among the often-used application programs supported by connectionless transport service (UDP) are the following:

PING network-level testing utility, which uses ICMP Echo and ICMP Timestamp messages to interrogate remote machines, collect statistics and produce reports (PING stands for Packet Inter-Net Groper).

XNET cross-net debugger, which is used for remote interactive loading, dumping and debugging, primarily for diskless machines.

Generic UDP server, including two name-address translation protocols, two time-service protocols a file-transfer protocol and a remote host monitoring protocol.

Utility programs for testing and debugging standard UDP services, including the above, as well as various surveying programs.

3. Fuzzball Applications

In following sections selected fuzzball applications are described, ranging from Internet gateway development through transport-level performance improvement to the development of efficient name-address translation mechanisms.

3.1. Internet Gateway Development

One of the earliest applications of Fuzzballs was as an Internet gateway, usually used between an experimental local net and the ARPANET. The first Internet gateways were implemented by Bolt, Beranek Newman (BBN) using PDP11 equipment, the BCPL language and the ELF operating system. Like the DCN system before it, ELF suffered from debilitating context-switching overhead and was limited to about 30 packets per second. The BBN implementation was eventually rehosted to the MOS operating system, which like the Fuzzball was built as a subset of its full-featured ancestor, and recoded in assembly code. This resulted in improved switching rates in the order of 200 packets/second.

Meanwhile, the Fuzzballs were being used as experimental development platforms for the Gateway-Gateway Protocol (GGP) used in the BBN system, and later for the Exterior Gateway Protocol (EGP) [13]. The goal of the experimental effort was to explore data-management techniques, protocol interoperability, routing algorithms and congestion control mechanisms. As the result of this exploration, the maximum switching rate of the Fuzzball (with LSI-11/73 processor) was improved to over 400 packets per second.

The participation of the Fuzzballs in the development of EGP was particularly important. EGP was developed in an atmosphere of controversy and delicate compromise between functionality and robustness. It is intended as both a primitive reachability protocol for use between administrative domains and a firewall to prevent instabilities within one domain from affecting another. In order to get the compromises just right, extensive analysis, prototyping and refinement cycles were necessary. Most of these involved Fuzzball development networks built on ARPANET paths, which required careful control of the routing environment and interoperation with existing GGP gateways. What made this practical was the easily butchered routing and forwarding mechanisms in the Fuzzball, in which firewall principles could be invented, explored and evaluated quickly and with low effort.

3.2. Internet Performance Issues

Of considerable interest in the early development and prototyping of TCP/IP were the issues of interoperability and performance. As various implementations of the protocol suite matured, in particular those for Multics, TOPS-20, Unix and Fuzzball, distributed testing sessions called bakeoffs were held. With the exception of the Fuzzball, these implementations were intended for long-term operational use, with operational parameters optimized for the traffic flows and rates typical for a multi-user timesharing system.

On the other hand, the Fuzzball implementation was specifically designed to explore the parameter space, sometimes in regimes ordinarily considered bizarre. For instance, Fuzzball utility hosts were configured with 1200-bps dial-up lines and used with TCP/IP and TOPS-20 hosts via Fuzzball gateways and ARPANET for ordinary virtual-terminal, file-transfer and electronic-mail applications. The purpose was not to establish an operational user environment, but to explore the practicality of the TCP/IP protocol suite and the various implementations at the extremes of the operational envelope.

The bakeoffs and other experimental programs demonstrated that TCP/IP could indeed work well with grossly mismatched systems and transmission media; however, the experiments also revealed that catastrophic performance degradation could occur unless careful attention were paid to certain aspects of the protocol implementation. Not surprisingly, the most serious problem was overrunning the gateway between the 56-Kbps ARPANET and the 1200-bps dial-up lines. The most sensitive areas in the implementation included the mechanisms for packetization, retransmission and acknowledgement generation.

In 1981 several experiments were carried out with the goal of exploring techniques to avoid unnecessary packet congestion in the gateways. These fell into three areas: reducing the incidence of small segments, especially in interactive virtual-terminal service, improving the efficiency of remote-echo and acknowledgement strategies and avoiding large pulses of data when large windows are first opened.

Techniques used by the Fuzzballs to reduce the incidence of small segments include send delays, in which data are held and aggregated until a sufficiently large segment has accumulated, aggregated retransmissions, in which data arriving since the first transmission are included in subsequent retransmissions, and adaptive thresholds which prevent transmission until the TCP window has opened to a respectable size. Techniques used to improve the piggybacking efficiency of acknowledgements include acknowledgement delays, which delay the acknowledgement until the received data have been copied to

user buffers and the application response (typically remote-echo data) are available.

Many of these techniques were employed in some (but not all) the implementations at the time, including the Fuzzball, and have been further refined since then. However, since the Fuzzball was operating at the extremes of the credible envelope, precise tuning of the dynamics was exceptionally critical, to the point where the parameters had to be adjusted on an adaptive basis, including the retransmission timeouts, packetization delays and acknowledgement delays. A particularly troublesome issue was the determination of retransmission timeout under conditions of moderate segment loss and path delays varying over several orders of magnitude. An extensive survey [12], conducted with the aid of Fuzzballs used as network sounders, revealed the Internet had path delays and delay dispersions much larger than imagined.

As the result of these experiments, several modifications to the suggested TCP retransmission-timeout estimation algorithm were made, including a nonlinear adjustment intended to improve performance under conditions of very high delay dispersion and provision for multiple per-segment range measurements to increase the sample density. In addition, an adaptive backoff algorithm was designed, along with explicit counting of outstanding segments with a rate-sensitive limit. These experiments used digital-analog converters and panel meters to watch critical time-varying variables, which resulted in some memorable demonstrations.

3.3. Applications Development

Several experimental Internet mail systems have come and gone during the lifetime of the Fuzzballs. As each was designed, implemented and evaluated, Fuzzball prototypes were used for test and debug of the protocol and various implementations. Perhaps the most ambitious of these projects involved the DARPA Multimedia Mail Project, in which research groups at SRI, BBN, ISI and Linkabit developed an experimental multimedia mail architecture and presentation protocol with real-time voice, image and text capabilities. The Fuzzball implementation includes special-purpose spooler daemons for digitized speech, facsimile and bitmap graphics, as well as rule-based mail sending, receiving and reading programs. It was used mostly to explore the features of the architecture, develop fast encoding and storage algorithms and verify interoperability of all implementations.

One of the most interesting experiments using the Fuzzballs involved the Domain Name System [17], which is now in widespread use as a host name/address directory service. The system is based on a distributed, hierarchical data base and a set of structured lookup procedures that can be used in connection-oriented or connectionless modes. The Fuzzball implementation includes both a domain-name server, which holds the data base, and a resolver subroutine, which is linked with every application program needing this service. A distinguishing feature of resolver subroutine relative to others implemented for Unix, is an amazingly persistent, adaptive search algorithm that painstakingly combs the data base of possibly many other servers in case of incomplete or inconsistent data.

4. Enduring Systems

Over the years Fuzzballs have been involved in enduring demonstrations and networks requiring commercial-grade operation, monitoring and control. Several of these applications, including two out of the three described in following sections, have provided service to large numbers of users under punishing conditions and have served to evaluate the performance of the Fuzzball implementation itself.

4.1. SATNET Demonstrations

The DARPA Packet Satellite Project was started in 1975 to adapt packet-switching technology to international satellite communications, resulting by 1979 in the Atlantic SATNET system which spanned the US, United Kingdom and Norway [6]. As part of an intensive measurement and evaluation effort, Fuzzballs were used as traffic generators, statistics collection platforms and general-purpose data reducers [3]. At the 1979 National Computer Conference, Fuzzballs were used in a demonstration of SATNET with real-time packet-voice conferencing, image transmission, and an experimental small-aperture Earth terminal [MIL80].

As additional countries such as Italy and the Federal Republic of Germany joined the SATNET community the Fuzzball was often used as a vehicle to gain familiarity with the SATNET technology and as a tool for experiments and measurements. At one time or another, the research and defense establishments of every SATNET participant had nests of Fuzzballs spliced to their local-net infrastructure and functioning as gateways, lightweight electronic-mail hosts and experimental platforms.

In order to support these activities a good deal of ad-hoc experimental software was developed, including data-reduction, graphics-display and format-conversion applications. The multiple virtual-process architecture and RT-11 emulator features of the Fuzzball proved highly useful and suited to this effort. Since the emulated RT-11 environment supports the FORTRAN, BASIC and C programming languages, as well as most RT-11 utility programs and text editors, real-time traffic generators,

statistics processors and display generators can be quickly constructed or adapted to particular experiments.

4.2. INTELPOST Network

The INTELPOST system was an electronic-mail network built in 1981 by BBN and COMSAT and operated by the US Postal Service and overseas affiliates. It linked sites in the US and Canada with several cities in Europe and South America and consisted of about a dozen PDP11-class processors with high-speed facsimile scanners and printers interconnected by serial lines with speeds to 56 Kbps. Each processor used the DEC RSX-11 operating system together with the Fuzzball TCP/IP networking software and Hellospeak routing algorithm. The software was adapted from the native Fuzzball code by simply replacing the macro library with another specifically tailored for the RSX-11 environment.

This system is believed to be the first commercial deployment of the TCP/IP protocol suite outside the research community. It operated for several years carrying revenue traffic, but remained largely a pilot project and unknown by most of the public. It was in this environment that early experiments in congestion control were carried out, including those based on flow modulation, which is described in another section. While the system used the standard TCP/IP protocol suite and addressing conventions and an experimental gateway was set up at COMSAT Laboratories, the only traffic ever exchanged between INTELPOST and the DARPA research community was monitoring and debugging packets and an occasional misrouted file.

4.3. NSFNET Backbone

The NSFNET is a loosely organized community of networks funded by the National Science Foundation to support the sharing of national scientific computing resources, data and information [7]. NSFNET consists of a large number of industry and academic campus and experimental networks, many of which are interconnected by a smaller number of regional and consortium networks. The NSFNET Backbone Network [15], which is the primary means of interconnection between the regional networks, includes switching nodes located at six supercomputer sites and three regional interties. Additional nodes are used for program development and testing, bringing the total to about thirteen.

A Backbone node consists of a Digital Equipment Corporation LSI-11/73 system with 512K bytes of memory, dual-diskette drive, Ethernet interface and serial interfaces. One or two low-speed serial-asynchronous interfaces are provided, as well as one to three high-speed serial-synchronous interfaces. All Backbone nodes include crystal-stabilized clock interfaces, while one node (NCAR) is equipped with a WWVB radio time-code receiver providing a network time reference accurate to the order of a millisecond.

The Backbone nodes function as Internet gateways [2] and include all the switching, monitoring, control and error-recovery functions necessary to forward packets from one regional Ethernet to another. Other gateways connect the Ethernets to regional and consortium networks, which in some cases span large regions of the country. Some sites are connected to other backbone networks such as ARPANET and public data networks as well.

In mid-June 1988 there were 5856 hosts on 918 networks interconnected by 236 gateways listed at the Department of Defense Network Information Center alone, which in itself is only a small fraction of the overall Internet. Of the approximately 460 nets now in regular operation, about 130 are either directly connected to the Backbone or indirectly by means of gateways and other regional and consortium networks. The aggregate backbone traffic has doubled over the last nine months to about 71 packets per second, or about 4.4 packets per second per 56-Kbps line.

5. Experiments

Among the many experiments mounted on the Fuzzball, three have been selected as typical and representing the scope and application of the Fuzzball implementation. The first examines issues in type-of-service routing and queueing, the second involves preemption and congestion management and the third describes an internetwork time service.

5.1. Type-of-Service and Precedence Queueing

The Internet protocol includes provisions for specifying type of service (TOS), precedence and other information useful to improve performance and system efficiency for various classes of traffic. The TOS specification determines whether the service metric is to be optimized on the basis of delay, throughput or reliability and is ordinarily interpreted as affecting the route selection and queueing discipline. The precedence specification determines the priority level used by the queueing discipline.

The Fuzzball routing mechanism includes provisions to determine the route based on TOS. The mechanism simply decodes the eight combinations of three TOS bits and appends the resulting octet to the four-octet Internet address. The routing tables use a mask-and-match scheme where the resulting 40 bits are first filtered by a mask specific to each entry, then matched against an address specific to that entry. The tables are ordered, with possibly several entries for the same address range, but different TOS masks, and with the first match found terminating the search. This scheme allows TOS interpretation to be integrated directly into the routing

mechanism and protocols; however, existing routing protocols, including Hellospeak, do not provide means to propagate TOS masks so entries using this feature have to be manually configured.

As the NSFNET Backbone has reached its capacity, various means have been incorporated to improve interactive service at the possible expense of deferrable (file-transfer and mail) service. An experimental priority-queueing discipline has been established based on the precedence specification. Queues are serviced in order of priority, with FIFO service within each priority level. However, many implementations lack the ability to provide meaningful values and insert them in this field. Accordingly, the Fuzzball cheats gloriously by impugning a precedence value of one in case the field is zero and the datagram belongs to a TCP session involving the virtual-terminal TELNET protocol.

The results of this scheme are mixed, as could be expected. Customers of the NSFNET Backbone were thrilled when TELNET response dramatically improved after the new scheme was installed. However, it sometimes happens that low-priority traffic is continually pushed back in the queue as the result of high-priority arrivals, only to be preempted, cause a quench message (see below) or simply to time out. However, if large quantities of deferrable data, such as file-transfers and mail, get pushed back in this way, quench messages tend to concentrate on the originating hosts and help dissipate the load.

5.2. Congestion and Preemption Strategies

In the TCP/IP protocol suite, which is based on end-to-end connectionless service, it is usually assumed that gateways have little state except that induced by the routing algorithm. In particular, the state of the various resources, including processor time, buffer space and queue contents, is generally invisible outside the gateway. This makes congestion avoidance and control very difficult, since the only access a congested gateway has to the state of any virtual flow is the composition of the queues in the gateway itself.

In order to deal with traffic surges, the Internet architecture specifies the ICMP Source Quench message, which is in effect a choke packet sent to the originating host by a downstream gateway when it experiences an overload. Some gateway implementations make an attempt to cope with congestion by emitting a quench when a datagram arrives for a queue whose size exceeds a threshold or when the number of datagrams dropped due to congestion-related reasons exceeds a threshold. Host implementations respond to quenches in various ways; some ignore it, some including (modified) Berkeley 4.3 Unix and Fuzzball, reduce the TCP window size by a multiplicative factor, after which each acknowledgment received causes an additive increase, and some, including Cray Research Unix, operate on rate-based principles such as described in [PRU86].

Experience suggests that such simple mechanisms have only marginal utility and then only for traffic surges where the delay on the path to the originating host is short compared to the period of the traffic surge. In the case of the NSFNET Backbone gateways, which is typical of other Internet gateway systems, congestion is characterized by frequent surges lasting up to a few minutes along with occasional intense bursts lasting only a few seconds. Bursts, often caused by misengineered TCP implementations, can be extremely damaging, since they tend to fill up queues quickly and are resistant to quenches.

Upon review of the extensive log information collected by the NSFNET Backbone Fuzzballs a key fact emerges: most congestion surges and bursts are caused by a relatively small number of originating hosts. This suggests an effective congestion policy should fairly distribute resources such as buffer space on the basis of originating host address. Congestion causes large queues, large delays, large delay dispersions and invites further abuse by undisciplined host retransmissions, even if heroic buffer space is available [18]. These observations suggest an effective congestion avoidance policy should be based on mean queue size and attempt to fairly equalize the sizes if more than one queue is present.

However, when a burst exceeds the buffer space available, the only defense possible is to either drop arriving packets or selectively preempt ones already queued for transmission. This naturally suggests the preemption policy should be an extension of the quench policy and result in preemption of those hosts which do not respond to quench. Furthermore, fairness in the form of equal access to system resources should be observed in order to avoid capture of excessive network resources by reckless hosts [15].

In an experiment designed to evaluate various realizations based on the above policies, the Fuzzball implementation was modified to incorporate comprehensive quench, preemption and service disciplines based on the specifications explicit and implicit in every IP datagram header. The result is a service model based on IP precedence and type-of-service, a fairness model based on IP source address and a congestion-control model based on mean queue size. This architecture can be summarized in the following rules:

1. Customers are identified on the basis of IP source address. Each distinct IP address is associated with a different customer for quench and preemption purposes.

2. Customer service classes are determined on the basis of IP precedence and type-of-service. Queues are serviced in order of class, with all customers of a higher class serviced in FIFO order before any customer of a lower class.

3. Every customer has equal claim to critical system resources, most importantly buffer space. In case of insufficient resources, the quench and preemption mechanisms operate to reduce the allocations of those customers claiming the most resources, so that the available resources will tend to be equally allocated among all customers.

4. The congestion state for each queue is determined by its mean size. Below a selected threshold no quench messages are sent. Above the threshold, quench messages are sent to customers determined by (3) above and at a rate proportional to mean queue size.

5. If upon a new arrival insufficient system resources (e.g. input buffers) can be found for subsequent arrivals, customers determined by (3) above (which may include the new arrival) are preempted from the queues until these resources can be found.

6. In case of ties when the above mechanisms would preempt any one of two or more queues, the queue selected is the one going longest since the last preemption.

In order to implement these mechanisms without significant performance penalty, it is necessary to minimize per-packet processing and minimize queue scanning. The technique implemented in the Fuzzball was adapted using timestamp mechanisms already in place for other functions. An arriving packet buffer is timestamped and inserted on an output queue determined by the routing algorithm in FIFO order by service class. When the buffer is removed from the queue for output, the difference between the removal time and the timestamp represents the queueing time for the buffer and is used in conjunction with the time-to-live field in the Internet header to determine whether the packet has lived too long in the system and should be destroyed.

The queueing time can also be used to estimate the mean queue size as follows. The queueing times for successive packets are summed over an interval depending on the output line speed or service rate, currently about four times the maximum packet transmission time. At the end of each interval the sum is exponentially averaged into an accumulator with a selected weight, currently one-half, and the accumulator compared against a threshold. If the threshold is exceeded a quench is sent to the originating host with the largest total buffer space on the queue and the accumulator is forced to zero.

This scheme is designed to avoid quenching hosts if the mean queue size is small, yet protect against excessive quench rates in case of overload. The averaging interval is chosen to match the expected mean path delay in the Internet, about a couple of seconds. The weight and threshold are chosen to begin triggering quenches when the mean queue size exceeds about one-half and reach maximum frequency (once quench per interval) when the mean queue size approaches two.

As reported previously [15] there is no doubt the selective preemption scheme is highly effective in reducing the impact of bursts due to misengineered host implementations or profound speed mismatch at the entry points to the NSFNET Backbone. Some idea of the effectiveness of the quench scheme is evident from the following: In July 1988 before quench was implemented, the NSFNET Backbone aggregate traffic load per queue ranged from 0.3 to 4.0 packets per second, with a mean of 2.0. At that time the preemption rate was .06 and timeout rate .03 percent. In March 1988, several months after quench was installed, the load ranged from 2.6 to 9.2 packets per second, with a mean of 4.7. At that time the preemption rate was 0.37, timeout rate 0.11 and quench rate 0.27 percent. The traffic on the sixteen 56-Kbps internodal trunks had doubled from 31.5 to over 71 packets per second, but the aggregate loss rate was still well below the system objective of one percent.

Tests with the Cray TCP/IP implementation show that congestion on NSFNET Backbone paths where the flow enters the system via an Ethernet are effectively throttled by the quench mechanism. Tests with a recently improved Unix 4.3bsd implementation show mean queue size reductions of about one-third over the uncontrolled size when quench is used. These data alone are insufficient to estimate the effect of the quench policy.

At this time few definitive conclusions can be reached on the effectiveness of the quench and preemption schemes with respect to the global environment, since they are implemented only in the Fuzzball (although some Fuzzballs are in busy places) and responsive host implementations are far from ubiquitous. While the experiment demonstrated that the schemes can be effective, additional work is necessary to determine the parameters and their affect on the overall system performance. Finally, the fact that quenches are generated at intervals depending on mean queue size, not at regular intervals or as the result of new arrivals, suggests that host response mechanisms based on flow estimators may work much better than those based on window-size estimators with these schemes.

5.3. Time Synchronization

Experiments conducted since 1981 have demonstrated that the Fuzzball implementation is ideally suited as an

internet time server. The Fuzzball local clock is constructed as a first-order phase-lock loop, in which measured offsets between the local clock and an external reference source are used to adjust the local-clock phase and frequency. The implementation is carefully constructed to minimize sources of jitter, such as interrupt latencies and resource conflicts. The resulting accuracy at the application interface is usually less than a millisecond in phase and less than a part per million in frequency.

A time-synchronization function is built into the DCN routing protocol [12], which uses a variant of the distributed Bellman-Ford algorithm [1]. One or more DCN hosts synchronize to an external reference source, such as a radio clock or time daemon, and the routing protocol constructs a minimum-delay spanning tree rooted on these hosts. The clock offsets along the tree are then computed using timestamps included in routing-update messages. Typical local-clock accuracies using Fuzzballs and the DCN routing protocol connected over LAN paths with serial lines and Ethernets are in the order of a few milliseconds.

In 1985 after an extensive set of experiments and prototype refinement using Fuzzballs at several locations in the US and Europe, an architecture model and protocol based on the DCN design, but suitable for use as a ubiquitous Internet time service, was proposed. The purpose of the protocol, called the Network Time Protocol (NTP), is to connect a number of master clocks, synchronized to national standards by wire or radio, to widely accessible resources such as backbone gateways. These gateways, acting as primary time servers, use NTP between them to cross-check the clocks and mitigate errors due to equipment or propagation failures. While multiple primary servers may exist, there is no requirement for an election protocol, such as used in some Unix systems [5].

In order to reduce the protocol overhead, some number of local-net hosts or gateways, acting as secondary time servers, can run NTP with one or more primary servers, then redistribute time to the remaining local-net hosts using NTP, DCN or some other protocol such as described in [9] and [5]. In the interest of reliability, selected local-net hosts can be equipped with less accurate but less expensive backup clocks and used in case of failure of the primary and/or secondary servers or communication paths between them.

The Fuzzball implementation includes algorithms for deglitching and smoothing clock-offset samples collected on a continuous basis, as well as algorithms for selecting good clocks from a population possibly including broken ones. These algorithms were evolved under typical operating conditions over the last two years. A comprehensive description of the NTP architecture, protocol and algorithms is given in [16].

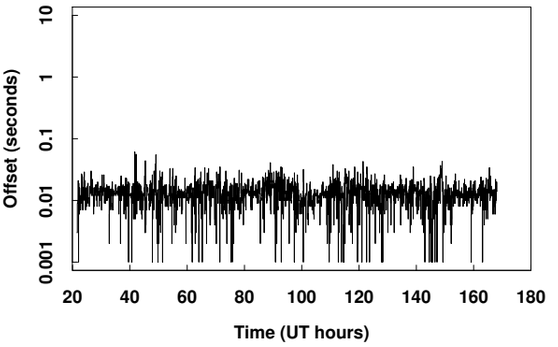

Figure 1. Measured Clock Offsets

Timekeeping accuracies achieved with the Fuzzball implementation on typical Internet paths are in the range of a few tens of milliseconds. Figure 1 shows the NTP clock offsets (absolute) measured between Fuzzball primary servers at the University of Delaware and University of Maryland over a period of about a week. Performance data collected with the primary servers show that this accuracy can be reliably maintained throughout most portions of the Internet, even in cases of failure or disruption of clocks, servers or communication paths.

There are presently six Fuzzball primary time servers located on the east coast, west coast and midcontinent, each of which serves 20-40 Fuzzball and Unix secondary time servers in the US and Europe. A survey reported in [16] suggests there may be well over 2000 potential clients in the ARPANET/MILNET community, plus probably several thousand more in the NSFNET community.

6. Parting Shots

During the past decade when the Fuzzball has been most useful, it represented a relatively inexpensive way to pry into network technology, prototype new architectures and protocols and in general fan the fuzes of the exploding Internet. As the cost of sophisticated workstations has plummeted in recent years, the LSI-11 technology itself is no longer economically attractive. The software, having been lashed together by several generations of programmers and in support of novel applications and ad-hoc experiments lasting over a decade, is not readily portable. For awhile, at least, Fuzzballs may lurk in dark corners of various laboratories in the hulks of otherwise outmoded PDP11 and LSI-11 systems and occasionally barge out to resolve routing problems and rescue wandering packets. In the long term Fuzzballs may endure as time servers and ad-hoc intelligence platforms, for which they are well suited.

A new generation of Fuzzballs may in fact have already happened in the form of recent Unix-based workstations, which are fast becoming as ubiquitous as the common video terminal. However, the casual abuse heaped on the Fuzzball operating system is much harder to do in these workstations, since much of the code is licensable and sources are not commonly available. In addition, the Unix kernel is considerably less tractable than the Fuzzball and much of the networking code is not easily modifiable. Nevertheless, the Unix programming support is far superior to the Fuzzball and much more portable. The ideal next-generation Fuzzball would be programmed in C, support the Unix run-time environment, TCP/IP and ISO protocol suites and contain no licensed code.

7. References

1. Bertsekas, D., and R. Gallager. *Data Networks.* Prentice-Hall, Englewood Cliffs, NJ, 1987.

2. Braden, R. Requirements for Internet gateways. DARPA Network Working Group Report RFC-1009, USC Information Sciences Institute, June 1987.

3. Chu, W.W., D.L. Mills, et al. Experimental results on the packet satellite network. *Proc. National Telecommunications Conference* (November 1979), 45.4.1-45.4.12.

4. Defense Communications Agency. *DDN Protocol Handbook.* NIC-50004, NIC-50005, NIC-50006, (three volumes), SRI International, December 1985.

5. Gusella, R., and S. Zatti. The Berkeley UNIX 4.3BSD time synchronization protocol: protocol specification. Technical Report UCB/CSD 85/250, University of California, Berkeley, June 1985.

6. Jacobs, I.M., R. Binder, and E.V. Hoversten. General purpose packet satellite networks. *Proc. IEEE 66, 11* (Nov 1978), 1448-1467.

7. Jennings, D.M., L.H. Landweber, I.H. Fuchs, D.J. Farber and W.R. Adrion. Computer networks for scientists. *Science 231* (28 February 1986), 943-950.

8. Leiner, B., J. Postel, R. Cole and D. Mills. The DARPA Internet protocol suite. *Proc. INFOCOM 85 (*Washington, DC, March 1985). Also in: *IEEE Communications Magazine* (March 1985).

9. Marzullo, K., and S. Owicki. Maintaining the time in a distributed system. *ACM Operating Systems Review 19, 3* (July 1985), 44-54.

10. Mills, D.L. An overview of the Distributed Computer Network. *Proc. AFIPS 1976 National Computer Conference* (New York, NY, June 1976).

11. Mills, D.L. Internetworking and the Atlantic SATNET. *Proc. National Electronics Conference* (October 1981), 378-383.

12. Mills, D.L. DCN local-network protocols. DARPA Network Working Group Report RFC-891, M/A-COM Linkabit, December 1983.

13. Mills, D.L. Exterior Gateway Protocol formal specification. DARPA Network Working Group Report RFC-904, M/A-COM Linkabit, April 1984.

14. Mills, D.L. Internet Delay Experiments. DARPA Network Working Group Report RFC-889, M/A-COM Linkabit, December 1983.

15. Mills, D.L., and H. Braun. The NSFNET Backbone Network. *Proc. ACM SIGCOMM 87 Symposium* (Stoweflake, VT, August 1987), 191-196.

16. Mills, D.L. Network Time Protocol (Version 1) specification and implementation. Electrical Engineering Department Report 88-04-01, University of Delaware, April 1988.

17. Mocapetris, P. Domain names - implementations and specifications. DARPA Network Working Group Report RFC-1035, USC Information Sciences Institute, November 1987.

18. Nagle, J. On packet switches with infinite storage. DARPA Network Working Group Report RFC-970, Ford Aerospace, December 1985.

19. Prue, W., and J. Postel. Something a host could do with source quench: the source quench introduced delay (SQuID), DARPA Network Working Group Report RFC-1016, USC Information Sciences Institute, July 1987.