Firewall Performance Analysis Report

10 August 1995

Chris Kostick
Matt Mancuso

CSC
Computer Sciences Corporation
Secure Systems Center — Network Security Department
1340 Ashton Rd., Suite E
Hanover, MD 21076
(410) 850-5411

Table of Contents

List of Figures

List of Tables

1. Executive Summary

The customer is in the process of a phased proof of concept deployment of firewall technology to provide protected Internet connectivity to their organizational infrastructure. Phase 1 of this effort involves: (a) the integration, installation, and set-up of a GAUNTLET firewall in a stand-alone laboratory environment with simulated outside (Internet) and inside (organizational) WANs/LANs; (b) the transition of the firewall's outside connectivity from simulated to actual Internet connectivity; and (c) the transition from a simulated inside organizational LAN to a live connection to the organization's IS Building as a trial deployment. As a part of the firewall deployment, we conducted performance analysis testing to evaluate the throughput and processing capacity of the firewall.

Cursory pre-testing revealed that the firewall could not support the desired connection or data throughput requirements. In fact, certain system and proxy parameters actually facilitated successful denial-of-service attacks against the firewall. Research into the root causes of these failures mandated changes in the firewall kernel configuration, system, disk partition, and proxy parameters. These new tuning parameters are required for proper and secure operation of the firewall and are outlined in Section 2, Initial Firewall Configuration Parameters. Once the firewall was re-configured and able to support performance testing, we proceeded into the testing phase which was broken up into several parts:

- Test environment;
- Test scenarios;
- Test results and analysis; and
- Conclusions.

The performance testing focused on the firewall's ability to handle multiple simultaneous connections and measured performance degradation when handling large amounts of data. The firewall accomplished all of the tests successfully.

The performance analysis revealed that, in some cases, the firewall performance became less than acceptable:

- Large numbers of simultaneous FTP transfers take an extraordinarily long time to complete as compared to doing one transfer.
- Multiple concurrent connections, or multiple large data transfers using HTTP, degrades the firewall seriously enough that it has a difficult time performing other tasks.
- A large number of permission rules (5,000-10,000) in the */usr/local/etc/netperm-table*, in conjunction with a high number of concurrent connections causes a significant delay in process setup time.

CSC evaluated the firewall performance problems and explored methods of mitigating the deficiencies. Our recommendations are included as Section 7, Performance Enhancement Recommendations.

Overall, the firewall performed at acceptable levels. The hardware and software configurations chosen for the firewall appear to be sound choices because of their highly configurable nature. The tests were representative of high-end network conditions and outside of the normal traffic load expected. Therefore, we believe the firewall implementation chosen will meet the initial needs of the organization’s Internet infrastructure.

2. Initial Firewall Configuration Parameters

The firewall will, potentially, be supporting many connections through it. The BSD operating system, as delivered, is configured to handle only a moderate number of processes and connections. The kernel needed to be modified to support a larger process space and also to allocate more memory for network information. In addition, in order to maintain a high number of concurrent connections, several of the application proxies needed to be modified and recompiled.

The modifications had to be done to even begin conducting the tests. It is felt by CSC that these modifications are mandatory for the organization's firewall.

2.1 User Processes

The kernel derives the maximum number of processes it can run by the kernel variable MAXUSERS defined in its configuration file. The formula used for the maximum process table entries is 10+16*MAXUSERS. To adjust this, the file */sys/i386/sys/conf/GAUNTLET-V30* was modified and the MAXUSERS line changed to read:

```
MAXUSERS        256
```

This configured the kernel to maintain at most 4106 processes.

2.2 Network Parameters

Two options were added to the kernel configuration file in order to increase the firewall's ability to accept and service many simultaneous connections. Added to the configuration file */sys/i386/sys/conf/GAUNTLET-V30* were the following lines:

```
options         "SOMAXCONN=128"
options         "NMBCLUSTERS=2048"
```

The options increased the amount of outstanding connection requests the firewall can queue for service and the maximum amount of network mbuf (memory buffer) clusters. The total amount of memory allocated by the kernel for management of network connections is NMBCLUSTERS*2048. The above will allocate 4MB of memory for network information usage. This number can, of course, be increased.

2.3 Remaking the Kernel

After the kernel configuration adjustments were made, a new kernel was compiled following the steps below:

```
# cd /sys/i386/conf
# config GAUNTLET-V30
# cd /sys/compile/GAUNTLET-V30
# make clean
# make depend
```

```
# make
```

When the compilation was finished, the new kernel was installed. The existing kernel was saved to a backup copy using the following steps:

```
# cp /bsd /bsd.orig
# mv bsd /bsd
```

The system was rebooted. If the new kernel had not worked, the original kernel could have been restarted by interrupting the boot process and booting from the saved kernel using the following command:

```
Boot: sd(1,0,0,0)bsd.orig -w
```

Detailed instructions, and more information on all of the tunable kernel parameters, can be found in the document *Building Kernels on BSD/OS Version 2.0*, which can be accessed by the command line:

```
% man -m bsdi config
```

2.4 Proxy Source Modification

The proxy processes that run on the firewall needed to be modified so that they were able to handle incoming connections at a relatively high rate. The installed implementation of the firewall proxies has limitations in the number of outstanding connections that can be queued. Not only should the kernel be modified (the SOMAXCONN option), but the proxy applications need to be modified, as well. The proxies identified for source modification are `tn-gw`, `rlogin-gw`, `ftp-gw`, and `httpd-gw`.

The source files for the proxy programs are located in */usr/local/src/fwtk*. Table 1 shows the proxys that need modification, the directory location of each, and the specific files and line numbers that need to be changed.

Proxy	Directory	Source Files	Line Number
tn-gw	/usr/local/src/fwtk/tn-gw	tn-gw.c	1575
rlogin-gw	/usr/local/src/fwtk/rlogin-gw	rlogin-gw.c	1361
ftp-gw	/usr/local/src/fwtk/ftp-gw	ftp-gw.c	1111
			1735
http-gw	/usr/local/src/fwtk/http-gw	http-gw.c	2303
		htmain.c	742
		ftp.c	242

Table 1. GAUNTLET Source Code Modification

These modification are simple and involve only changing an argument to the `listen()` system call in all of the sources. The `listen()` call's second argument specifies for the program, the number of outstanding connection requests that will be queued. The recommended number for this argument is the SOMAXCONN option value, as defined in the kernel configuration file. For our purposes, that number is normally 128. Figure 1 shows the "before" and "after" modifications of the *tn-gw.c* source file. All of the other proxies need to be modified in the same way.

(before)

```
          ...
     if(listen(x,1) < 0) {
          perror("listen");
          exit(1);
     }
          ...
```

(after)

```
          ...
     if(listen(x,128) < 0) {
          perror("listen");
          exit(1);
     }
          ...
```

Figure 1. Proxy Source Code Change to the `listen()` Call

2.5 /etc/inetd.conf Modification

The source code changes had to be made because the proxies could not rely on inetd to start them fast enough. If inetd had too many outstanding connection requests coming in, it would shut down the service. Inetd could only queue up to 40 requests before all of the others started receiving "Connection Refused" messages. To this end, the *inetd.conf* needs to only listen for two services — authsrv and auth. Also, the "rje" service can be included for remote administration if desired. All of the other services the firewall maintains will be started in */etc/rc.local*. Figure 2 illustrates the contents of the *inetd.conf* and Figure 2. */etc/inetd.conf* File
shows the portion of the */etc/rc.local* file that starts the other proxy services.

```
# Internet server configuration database
#
#     @(#)inetd.conf    5.4 (Berkely) 6/30/90
#
rje      stream  tcp  nowait  root   /usr/local/etc/netacl   telnetd
authsrv  stream  tcp  nowait  root   /usr/local/etc/authsrv  authsrv
auth     stream  tcp  nowait  nobody /usr/local/etc/identd   identd
```

Figure 2. */etc/inetd.conf* File

```
/usr/local/etc/smapd
/usr/local/etc/mqueue &
/usr/local/etc/smap -daemon &
/usr/local/etc/tn-gw -daemon &
/usr/local/etc/rlogin-gw -daemon &
```

```
/usr/local/etc/ftp-gw -daemon ftp ftp-gw &
/usr/local/etc/http-gw -daemon 80 &
/usr/local/etc/http-gw -daemon 70 &
```

Figure 3. */etc/rc.local* Proxy Startup

3. Test Environment

3.1 Lab Configuration

The performance testing was conducted in the CSC laboratory in Hanover, MD, using the configuration shown in Figure 4.

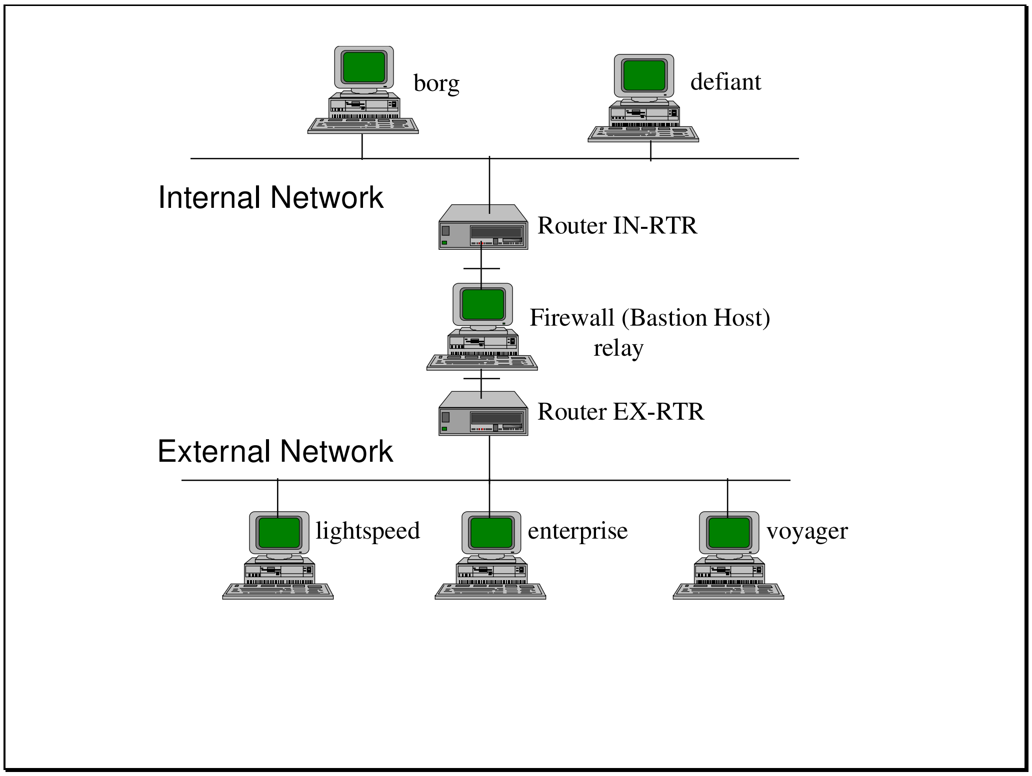

Figure 4. Laboratory Configuration

The architecture of each machine is specified in Table 2.

Machine Name	Architecture	Operating System	Memory
IN-RTR, EX-RTR	Wellfleet ASN	ROM 8.0.1	
relay	Compaq Proliant 1500	BSD 2.0	208M
borg	Sun 4/60	SunOS 4.1.3_U1	16M
defiant	Sun Sparcstation 1+	SunOS 4.1.3_U1	32M
voyager	Sun Sparcstation 1+	Solaris 2.4	32M
lightspeed	DECstation 5000/240	Ultrix 4.4	48M
enterprise	NeXTstation	NextStep (Mach 2.0)	20M

Table 2. Test Lab Machine Architectures

3.2 Performance Analysis Methodology

The performance analysis of the firewall consisted of several tests to measure the possible throughput degradation of well-used services. The primary services considered were TELNET, FTP, HTTP, and E-mail. Other services which will be going through the firewall, such as DNS, were not considered because of their relatively light traffic load.

The testing environment has many variables involved with evaluating the performance of the firewall: connection setup time on the initiating host, connect setup time of the receiving host, the number of concurrent processes on the each of the testing machines, and the bandwidth of the intermediate networks.

The bandwidth of the networks was not an immediate factor in the lab tests. The end-to-end connection consisted of ethernet networks with a transmission rate of 10 Mbps. For the initial installation in the organization's network, the firewall will sit behind a 56 Kbps connection to the Internet; therefore, these tests show performance of the firewall under high throughput network connections. The test conditions were set up such that the networks were not active with other traffic at the time. The hosts only had to compete with each other for the bandwidth of the ethernet.

The variables involving the initiating and receiving machines on either side of the firewall play a part in the variances of some of the results. Nonetheless, these variances should be consistent and should not be a factor when considering of the overall performance of the firewall.

4. Test Scenarios

The performance tests were broken into two categories.

- Individual performance tests; and
- Netperm-table lookup evaluations.

4.1 Individual Performance Tests

The individual performance tests measure the relative degradation of each service to be active through the firewall. The services measured are TELNET, FTP, HTTP, and E-mail.

4.1.1 TELNET

TELNET sessions are used for remote terminal emulation over a network, essentially remote logins. While TELNET is considered a low-bandwidth service because of its interactive nature, it is a popular service and, therefore, can have a cumulative performance impact. The tests run for TELNET, however, were not designed for throughput considerations. Rather, the TELNET tests gave an indication of performance degradation of running multiple concurrent processes. The test was run in several scenarios:

- 1 connection;
- 128 concurrent connections; and
- 256 concurrent connections.

The single connection establishes a baseline for connection setup time. The 128 and 256 concurrent connections test any variations in setup time for progressively more connections to the `tn-gw` proxy process. Connections were established to the bastion host where they sat idle for four minutes, and were then released. Each connection was timed and the difference between the total time taken and the four minute idle time represented the connection setup time.

4.1.2 FTP

Ftp, the program, is an implementation of the File Transfer Protocol (FTP). File transfers are one of the more bandwidth-intensive applications on the Internet today. It is expected that a number of FTP sessions will be running simultaneously on the firewall through the `ftp-gw` proxy. The FTP tests focused on throughput considerations and were designed to measure the amount of throughput the `ftp-gw` proxy process can handle. The FTP tests were run under the scenarios listed in Table 3.

Number of Concurrent Connections	File Size
1	256K
	1M
	5M
	10M
64	256K
	1M
	5M
	10M
128	256K
	1M
	5M
	10M

Table 3. FTP Test Scenarios

4.1.3 HTTP

The hypertext transfer protocol (HTTP) is the protocol most widely used for World Wide Web (WWW) access. Applications using HTTP became the most bandwidth intensive programs on the Internet in 1994, surpassing even E-mail. Most environments that have direct Internet access have followed this pattern and the organization will probably be no different. The HTTP tests examined two different environments: high volume of connections and high volume of data. Table 4 describes the various performance tests for high volume of connections and Table 5 shows the test events for high volume of data.

Number of Concurrent Connections	HTML Document Size
1	50K[1]
256	50K
512	50K

Table 4. HTTP High Volume of Connections Test Scenarios

Number of Concurrent Connections	HTML Document Size
1	512K
	1M
	2M
64	512K
	1M

[1] Document size based on a rough average of the HTML documents found on 10 of the most popular Web sites on the Internet.

	2M
128	512K
	1M
	2M

Table 5. HTTP High Volume of Data Test Scenarios

4.1.4 E-mail

Electronic mail, or E-mail, was the most used application on the Internet since its inception over twenty years ago. It is the primary mechanism of communication for users on different networks other than the Internet, such as BITNET and UUCP networks, as well as on-line services, such as CompuServe, America On-Line, and Prodigy. It is estimated that 12 million people use E-mail over the Internet. E-mail will remain a mainstay of communications on the Internet for a while. organizational users will undoubtedly contribute to the vast amount of E-mail traversing the Internet everyday.

The E-mail performance tests were designed to evaluate the amount of E-mail the firewall can process and how well the firewall handles large E-mail messages. E-mail going into and out of the organization's network will go through the firewall. However, E-mail going 'through' is not the same as other services, such as TELNET or FTP. With FTP for example, the firewall establishes a connection through the `ftp-gw` proxy. E-mail is a store-and-forward service using the `smap` and `smapd` proxy processes. Mail is received, stored on the firewall, and later delivered to the internal mailhub. Therefore, E-mail test scenarios will differ from the preceding test scenarios. The queuing nature and the variability of sendmail V8 prohibit any accurate measurement of how 'long' it took to deliver the mail.

The E-mail tests, listed in Table 6, will yield a measurement of success, as opposed to a measurement of time. Snapshots of queue sizes will be taken and system utilization monitored. Message sizes were based on the files listed in Table 7.

No. of Messages (concurrent)	Message Size
512	10K
128	564K
128	1.2M

Table 6. E-mail Test Scenarios

File Size	Files Type
10K	ASCII Text
564K	uuencoded GIF file
1.2M	Postscript file

Table 7. Files Types Used in E-mail Tests

4.2 Netperm-table Lookup Tests

Whenever an application proxy is started to service a connection, it will read the contents of the */usr/local/etc/netperm-table* looking for any rules that apply to it. It will then determine, based on those rules, if it is allowed to service that connection and whether or not additional functions, such as logging and/or authentication, need to take place.

The netperm-table tests measure any performance degradation due to the reading of a large netperm-table from disk for every proxy. The tests conducted will be the same as the 256 TELNET connections test described in Section 4.1.1 with the addition of the netperm-table modifications as shown in Table 8.

No. of Rules for `tn-gw` proxy
500
1000
2000
5000
10000

Table 8. Netperm table Rule Count

5. Test Results and Analysis

5.1 TELNET

The TELNET tests were run under the scenarios described in Section 4.1.1. The firewall was able to establish all of the TELNET connections simultaneously.

For the 128 connections test, 64 were run from defiant (tests 1-64) and 64 were run from borg (tests 65-128). For the 256 connections test, 128 were run from defiant and 128 were run from borg. Appendix A contains the shell scripts used for generating the traffic.

5.1.1 Results

Table 9 shows the average times (in seconds) for connection establishment to the firewall. Figure 5 illustrates the connection times charted for both 128 and 256 simultaneous connections. Appendix B contains the raw data set for the measured times.

No. of Concurrent Connections	Minimum Connect Time	Maximum Connect Time	Average Connect Time
1[2]	0	1	0.233
128	0	6	1.266
256	0	4	1.023

Table 9. Connection Establishment Time (in seconds) to the `tn-gw` Proxy

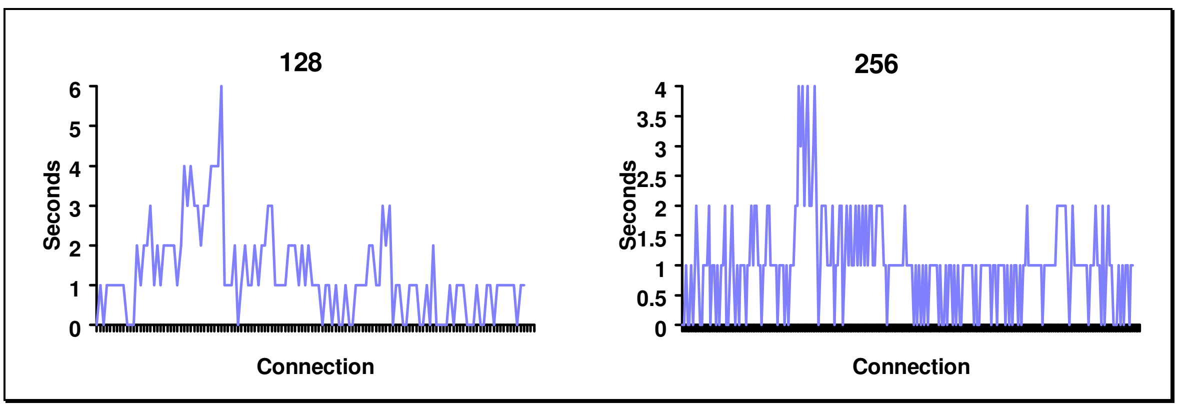

Figure 5. Connection Establishment Times for 128 and 256 Connections

5.1.2 Analysis

The firewall had no problem in maintaining connection establishment times at a reasonable level. And, as can be seen in Table 9, connection establishment was not

[2] Data collected as a series of individual connections one after another with no other connections established.

degraded as more `tn-gw` processes were started. On the contrary, time seemed to improve. However, because the time periods being measured were so short, it is hard to conclude that 256 simultaneous connections can be handled 'better' than 128. But it can be said that, with 256 connections, the level of service will be the same as 128. The amount of connections the firewall can handle scaled beyond the capacity with which the internal Sparcstations could generate.

5.2 FTP

The FTP tests were run under the scenarios described in Section 4.1.2. The firewall was able to perform all of the file transfers. Not all of the tests could be conducted, however, because the Sparcstations on the internal network were unable to generate the traffic load necessary for the 128 simultaneous 10M file transfer.

For the 64 connections test, 32 were run from defiant (tests 1-32) and 32 were run from borg (tests 33-64). Defiant and borg were the initiating machines where first a connection was made to voyager and a file retrieved, and then a connection was made to lightspeed and a file retrieved. The FTP processes were 'backgrounded' so that they all started at relatively the same time. For the 128 connections test, 64 were run from defiant (tests 1-64) and 64 were run from borg (tests 65-128) utilizing the same procedures as the 64 connections tests. Appendix A contains the shell scripts used to perform the tests.

5.2.1 Results

Table 10 shows the average times for completion of the file transfers. Figure 6 and Figure 7 illustrate the file transfer times for 64 and 128 concurrent connections, respectively. Appendix B contains the raw data sets for the measured times.

	No. of Concurrent Connections		
File Size	1	64	128
256K	1.87	42.19	100.50
1M	7.52	262.09	503.49
5M	37.5	1425.33	2786.86
10M	74.5	2866.88	—

Table 10. Average FTP Transfer Times (in seconds)

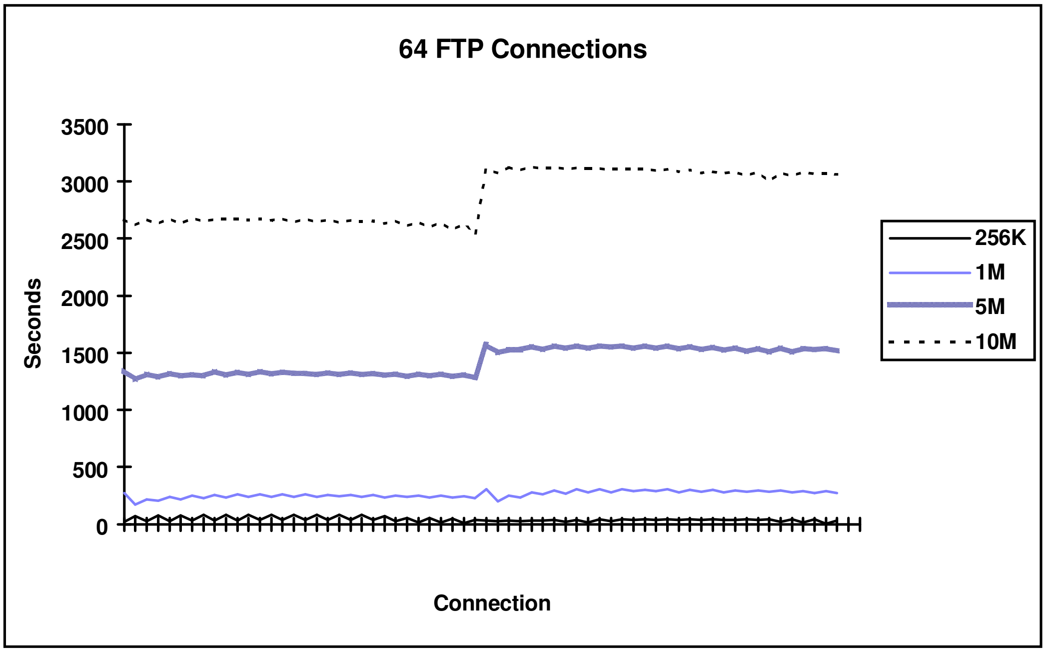

Figure 6. 64 Simultaneous FTP Connection Times

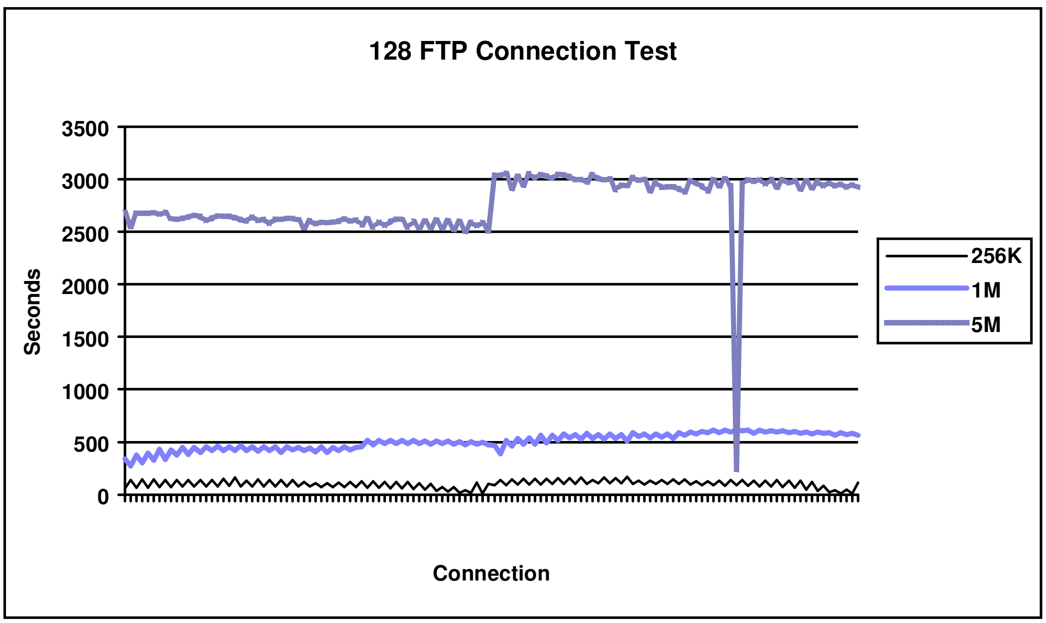

Figure 7. 128 Simultaneous FTP Connection Times

5.2.2 Analysis

Several items can be observed from the test results. First, as expected, the more connections going through the firewall, the slower the transfer times. It should be noted that all of the transfers were able to finish as far as the firewall was concerned. Test number 107 of the 128/5M transfer could not be completed. However, this was not due to the firewall. One of the test machines (borg) was unable to handle that one connection. That failure explains the sharp drop in the graph at that point. Similarly, the 128/10M tests could not be performed, again, however, not due to a failure of the firewall.

A second observation is that the time delay is not linear. This means that when doing 64 simultaneous connections, it did not take 64 times longer than one connection. The same was true for 128 connections. Nonetheless, the transfer times were not extraordinary.

The CPU utilization was monitored during all of the file transfers. The processor's idle time only went below 60% once (down to 59.1%) during the 128/5M transfer. This means that the firewall was relatively free to do other processing.

The graphs reveal some information about the variances of the sending and receiving hosts. Notice at the middle of the transfers that the graph takes a jump. This is because of the difference between defiant and borg. Borg, being slower and with less memory, had to do more swapping and, therefore, its transfer times were slower. Also, the chart lines have a relative up and down motion for the transfers. This is due to disparities in the disk access speeds of the sending hosts (lightspeed is much faster than voyager). Despite those disparities, the chart lines for the 5M and 10M transfer times are relatively level. This is due to BSD's load balancing of the `ftp-gw` processes.

5.3 HTTP

The HTTP tests were run under the scenarios described in section 0. The firewall was able to perform all of the HTTP GET protocol requests.

The machines lightspeed and voyager had httpd v1.4 with several HTML documents and GIF files available for retrieval. For the 256 connections test, 128 were run from defiant (tests 1-128) and 128 were run from borg (tests 129-256). Each machine used the lynx program to retrieve a specified document. The retrievals were 'backgrounded' so that they would all start at relatively the same time. The 512 connections test, as well as all of the high volume of data tests, followed the same scenario as the 256 connections test. The first half of the connections originated from defiant and the second half originated from borg. Appendix A contains the shell scripts used for the tests.

5.3.1 Results

Table 11 shows the average transfer times for retrieving a 50K document through HTTP. Figure 8 illustrates the transfers times for both 256 and 512 concurrent connections.

No. of Concurrent Connections	Average Transfer Time (seconds)
1	3.25
256	8.18
512	32.53

Table 11. HTTP High Volume of Connections Average Transfer Times

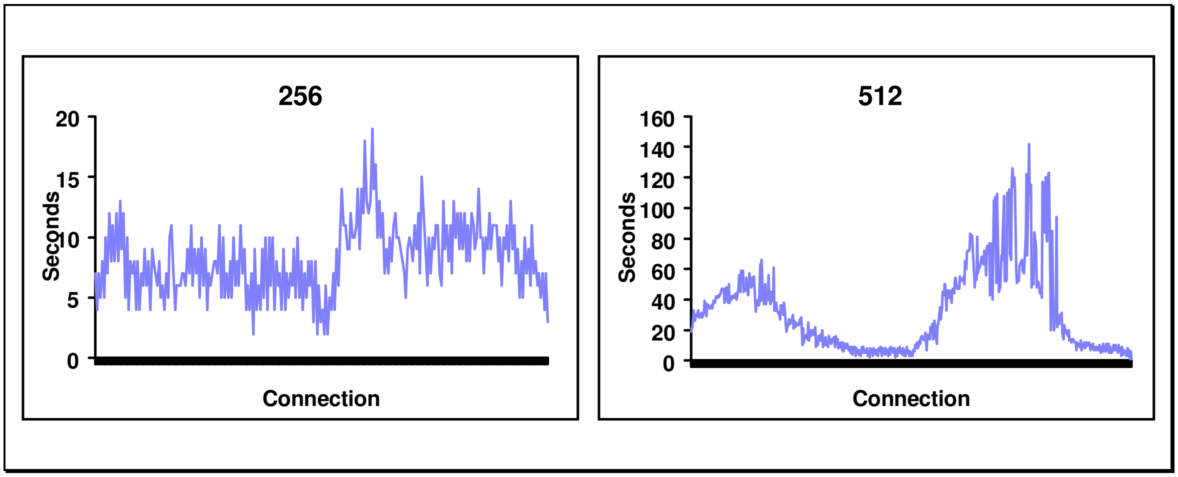

Figure 8. Transfer Times for 256 and 512 HTTP GET's of a 50K Document

Table 12 shows the average transfer times for the scenarios described in Section 0. Figure 9 illustrates the connections times for 64 simultaneous connections and Figure 10 illustrates the connection times for 128 simultaneous connections.

Document Size	*No. of Concurrent Connections*		
	1	64	128
512K	4.37	25.67	43.05
1M	5.28	80.67	104.92
2M	7.33	164.22	256.23

Table 12. Average HTTP Transfer Times (in seconds)

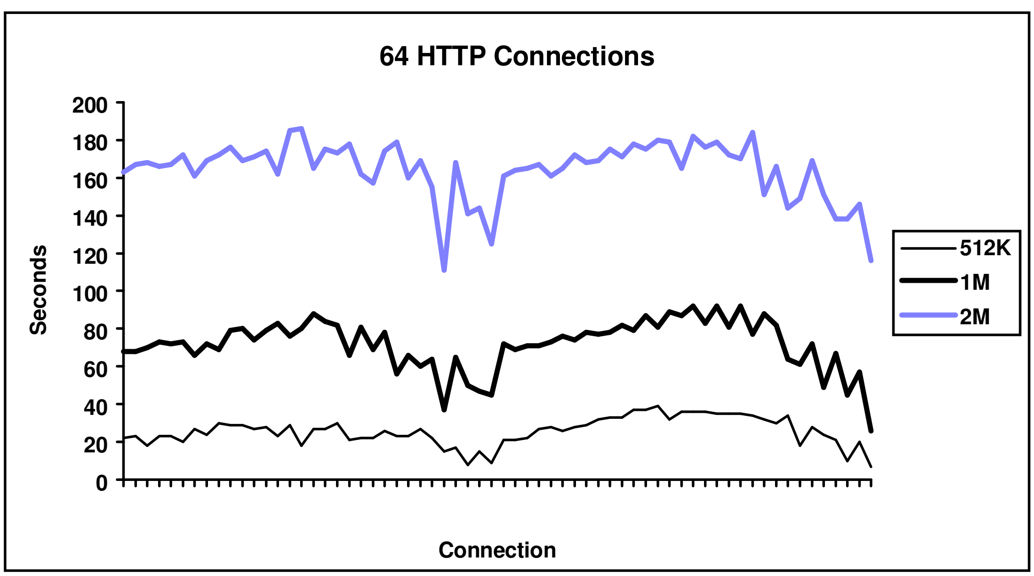

Figure 9. 64 Simultaneous HTTP Connection Times

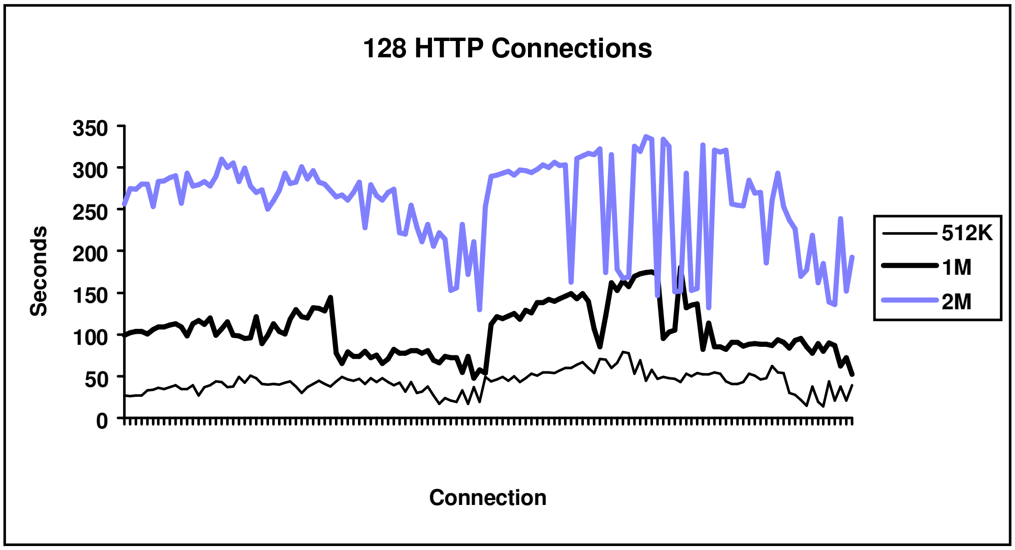

Figure 10. 128 Simultaneous HTTP Connection Times

5.3.2 Analysis

The HTTP tests showed a remarkable change compared to the TELNET and FTP tests. Several observations were made when sending HTTP traffic though the firewall.

By far the most important was that CPU utilization went to 100%, or 0% idle, both the high number of connections and high volume tests. The firewall could only accomplish one task — run the `http-gw` proxies. That was an incredible change compared to FTP where, doing the same transfer (128/1M), the CPU sat 60% idle. The relative idle time going to 0% is meaningless unless the load average of the system is also considered. The load average is an averaged count of the number of processes waiting in the run queue normally measured over the last 5, 10, and 15 seconds. On a system with little load, this number should be between 0.0 and 1.0. While the `http-gw` proxies were running, the system load average climbed to an astonishing 21.32.

It was also noted that BSD did not load balance the processes as well as it did for FTP. The times were sporadic, not consistent. However, the transfer times were much faster ($^1/_3$ - $^1/_5$ the time) than a comparable FTP transfer, but at a price of more processor time.

To further investigate the resource-intensive nature of the `http-gw` proxy, a source-code level examination was done. The `http-gw` proxy was profiled to see where the code was spending most of its time. This analysis revealed that the proxy spent a great deal of time executing the `read()` and `write()` system calls. This was to be expected since it was doing a lot of socket I/O. However, after looking at the network traffic of one of the HTTP transfers, it seemed it was doing far more than necessary. The same FTP transfer had $^1/_3$ less traffic on the network than HTTP. Another problem noticed with `http-gw` was that it was spending a lot of time in string function routines. This was traced to the behavior of the `http-gw` proxy parsing every request that it received from both the client and the external server sides. The parsing routines had a large impact on the CPU.

5.4 E-mail

The E-mail tests were run under the scenarios described in Section 4.1.4. The firewall was able to perform all of the mail deliveries required for the tests.

The machines lightspeed and enterprise were the sending machines. Mail messages were sent to the firewall, where it, in turn, delivered the mail to the internal mailhub (defiant). For the 256 deliveries test, 128 were run from lightspeed (tests 1-128) and 128 were run from enterprise (tests 129-256). Each machine used the sendmail program to send the mail message. The deliveries were 'backgrounded' so that they would all start at relatively the same time. For the 128/564K and 128/1.2M tests, 64 were run from lightspeed (tests 1-64) and 64 were run from enterprise (tests 65-128). Appendix A contains the shell scripts used for the tests.

5.4.1 Results

Table 13 shows the results of the E-mail tests described in Section 4.1.4.

Test	No. of Messages	Successful Delivery	Average Load	Max. Queued	Low Idle
10K Message Size	256	YES	2.43	114	44.1%
564K Message Size	128	YES	2.62	87	0.5%
1.2M Message Size	128	YES	1.25	61	0.0%

Table 13. E-mail Test Results

5.4.2 Analysis

The E-mail tests were all successful and posed no significant problem for the firewall. It should be noted the CPU idle times did go down to 0.5% and 0.0%. However, the load average stayed between 1.25 and 2.62. This indicates that, while the idle times went down to 0.0%, it happened infrequently. After monitoring which processes caused the high CPU utilization, it was determined that sendmail was the culprit. When `smap` or `smapd` ran, the CPU remained relatively free. Sendmail would not have caused a high CPU utilization on the firewall anyway. The program has an entry in the */etc/sendmail.cf* file that will cause it to queue any remaining mail for later if the load average goes above a certain threshold[3]. The option

```
Oxn
```

has a value of *n* that defines the threshold. The initial value of *n* was 4.

Since the `Ox` option is a tunable parameter and one of the basic functions of the firewall is to deliver mail, it is probably suitable to adjust this number to a larger value, such as 10 or 12.

---

[3] The calculation is actually a little more involved than just described, but for our purposes the given definition will suffice.

The queue sizes for the firewall did not remain at zero as would be expected since the load average did not go above four. The reason messages were queued was do to the mailhub receiving the messages. Sendmail has another option in the */etc/sendmail.cf* file that will cause sendmail to refuse more connections when the load average becomes higher than the defined value. The option is `OX`, and this turned out to be the case with defiant. It did not seem necessary to change this value for further testing because the messages were successfully delivered later by the firewall.

A problem that was noted during the E-mail tests was the auth service for the IDENT protocol was shutdown a couple of times. Sendmail V8 uses the IDENT protocol in an attempt to read the login name of the user who initiated the connection. This was once again traced back to inetd not being able to handle more than 40 simultaneous connections. A solution would be to run a modified `identd` process listening on the auth port and not have inetd service those connections.

5.5 Netperm-table Lookup Tests

The tests concerning the */usr/local/etc/netperm-table* were conducted according to the scenarios described in Section 4.2. The same scripts used for TELNET were used for the netperm-table tests and are included in Appendix A.

5.5.1 Results

Table 14 shows the average connection setup times for 256 connections in conjunction with the additional `tn-gw` rules. As a baseline, the average connect time for a single connection is also given. Appendix B contains the raw data sets for the measured times.

No. of Rules	Avg. Connect Time (secs) 1 Connection	Avg. Connect Time (secs) 256 Connections	netperm-table File Size
500	0.23	1.37	23K
1000	0.50	1.25	37K
2000	0.75	1.93	63K
5000	1	16.22	156K
10000	1	156.03	248K

Table 14. Average Connection Time to `tn-gw` Proxy

5.5.2 Analysis

Obviously, there was an impact when more rules were added to the netperm-table. A significant jump was noticed between 2000 and 5000 rules. However, disk access was not the problem. The proxy reads the netperm-table every time it starts. But, since BSD caches file accesses, file reads were from memory and not from disk. The problem stemmed from the fact that the file being searched in memory was being accessed by 256 processes at the same time. Apparently, the BSD operating system is not able to handle this concurrent access situation with speed.

We initially postulated that increasing the disk cache by adjusting the kernel parameter BUFMEM might improve the system's access time. However, BSD, by default, uses 10% of physical memory for disk caching. For the firewall machine, this turned out to be 21 Mb, which was more than enough.

6. Conclusions

The network topology for the experiments was end-to-end ethernet connectivity, and the firewall functioned exceptionally well. The “live” connection in the customer’s environment will be a 56 Kbps line to the Internet. This will be the bottleneck, not the firewall. Because the firewall is at the point of the bottleneck, it is often blamed for performance problems, when actually lack of bandwidth is the root cause.

The firewall performed well, even under stressful conditions, only after the kernel was reconfigured and the application proxies were modified to handle a larger queue size for connections waiting to be accepted. At no time did the firewall fail to complete a task once the modifications were made. This is not to say there were not problems.

The firewall handled large amounts of traffic through FTP well. However, it suffered a great deal with moderate to large amounts of HTTP traffic. Even though the FTP transfers were handled smoothly, the transfer times became increasing large to the point where it took 47 minutes to transfer a 10 MB file.

The netperm-table lookup tests showed the proxy connection setup time became intolerable when using a large number of rules in conjunction with multiple concurrent connections. The threshold where setup time approached a level of unacceptability was near 5000 rules for 256 simultaneous connections. This should not be viewed as a significant problem, however. Even for an environment as large as the customer’s, 5000 rules applied to one service is far beyond what will actually be used.

E-mail never appeared to be a problem. When the system load became too high on either the sending or receiving machines, the mail was simply queued for later delivery.

Overall, the hardware and software chosen for the firewall implementation seems to be a sound choice from a performance perspective. The BSD operating system is flexible and highly configurable. The GAUNTLET toolkit provides the source code to the proxies and this allowed for convenient modification to the applications, when needed. The Compaq Prolinea has plenty of memory, a fast CPU, and more than enough disk space. The performance tests executed for this report are representative of very high-end network conditions. Those conditions exceed the customer’s normal environment. Therefore, it is expected the firewall will meet the needs of the customer.

7. Performance Enhancement Recommendations

The firewall is one component of the overall connectivity architecture. Other mechanisms and network topologies can be implemented to increase the performance of the firewall. Outlined in this section are a few recommendations to help improve the entire firewall structure.

7.1 Filesystem Partitioning

The Compaq machine is configured with 12 GB of disk space - two 2 GB drives and two 4 GB drives. Initially, only one drive is configured for firewall usage, one of the 2 GB drives. Table 15 describes how the filesystems should be setup.

Drive	Size	Partition(s)
sd0	2 GB	/, /usr
sd1	2 GB	/usr/local
sd2	4 GB	/var
sd3	4 GB	/var/log

Table 15. Filesystem Configuration

The key is to put */var* and */var/log* on their own partitions. This allows substantial disk space for logging (*/var/log*) and for queuing mail (*/var*).

Larger filesystem capacity will allow the firewall to maintain more logs for auditing purposes and also be able to accept and queue more mail. A larger mail queue takes the burden of possibly queuing the mail for later delivery off the originating machine, thus saving network bandwidth.

7.2 Network Topology

At the very least, a T1 line (1.544 Mbps) should be the connectivity to the Internet. However, for the size and expected amount of traffic from the internal network, that may still prove to be a bottleneck. Other network architectures should be considered in order to gain maximum performance from the firewall. Two are suggested below.

7.2.1 Multiple Firewalls / Multiple Connections

A multiple firewall configuration is illustrated in Figure 11. This architecture statically balances services by offering dedicated, *separate* paths through the firewalls to the Internet. As an example, relay1 would handle TELNETs, FTPs, DNS, E-mail, etc. Relay2 would be the HTTP proxy server and, probably, the secondary DNS.

This type of architecture also provides good reliability. To continue the example, relay2 can be setup to provide TELNET, FTP, and E-mail, but only when relay1 is detected as no longer functioning. Configuring the firewall to perform this function is a little more than trivial, but possible.

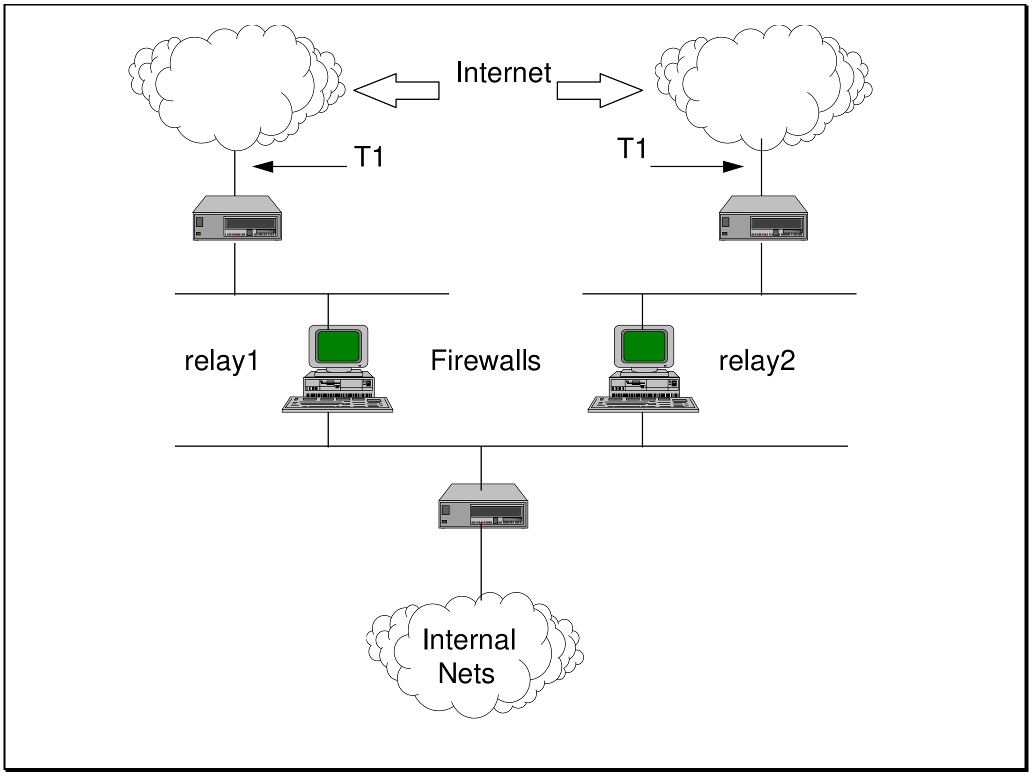

Figure 11. Multiple Firewall Configuration

7.2.2 T3 Connection

The high-end solution would be to have a T3 connection (45 Mbps) from the Internet provider. Obviously, there is a substantial cost factor involved with this architecture. The bottleneck at this point could be the ethernet segments on either side of the firewall. A better architecture would be the one illustrated in Figure 12, where 100 Mbps LAN segments would sit on either side of the firewall leading into the internal networks infrastructure. The LAN segments could either be 100 Mbps FastEthernet or FDDI fiber rings. BSDI will have network drivers for both network architectures available in early September, 1995.

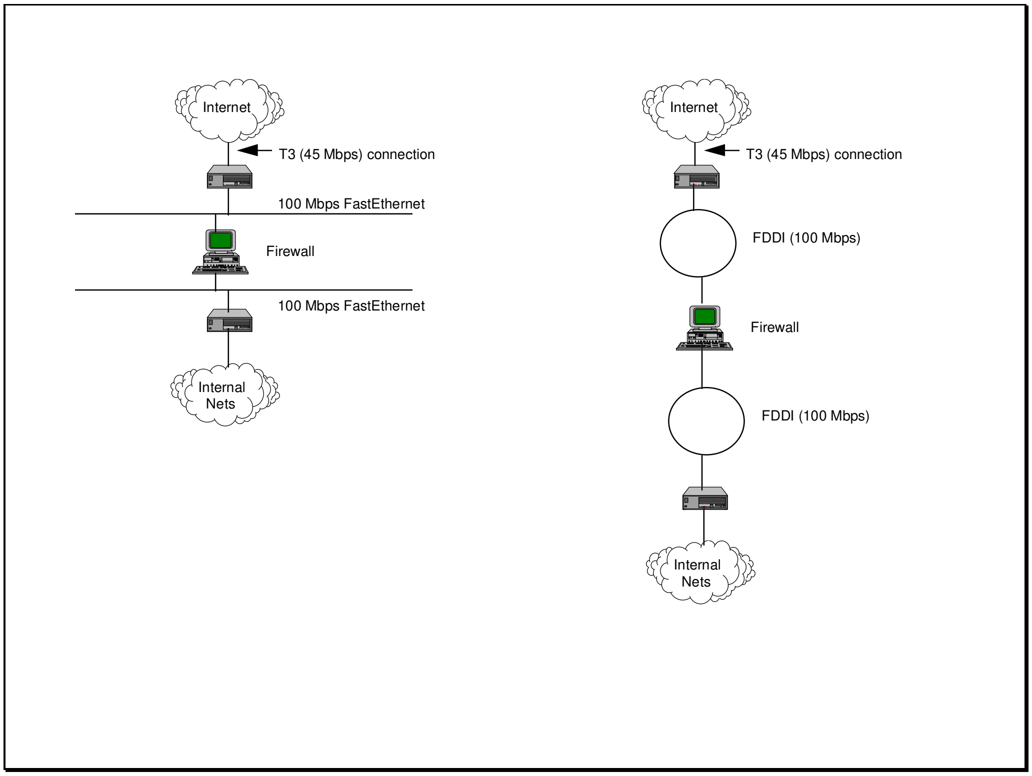

Figure 12. T3 Connection with 100 Mbps LAN Segments

7.3 HTTP Caching Proxies

Because HTTP is such a resource intensive application and causes hard performance hits on the firewall, special considerations should be given to its use. The goal is to have as little HTTP traffic as possible going through the firewall while still maintaining full service to the Internet. Fortunately, for HTTP, there is a solution — a caching proxy. Essentially, what would be created is a double-proxy environment. Figure 13 illustrates the architecture used for this proxy-within-a-proxy environment.

The flow of information first starts at the client making a request to the caching proxy (1), instead of the firewall. The caching proxy will examine the URL and, if the request had not been made earlier in the day, it will forward the request to the firewall (2). The firewall will obtain the information from the Internet (3)(4) and send the reply back to the caching proxy (5). The caching proxy will store the URL source information on disk and send the final reply back to the client (6). If a second request is made for the same URL, or another user on the internal network wishes to retrieve the same information, the caching proxy will already have the file and be able to respond directly skipping steps (2), (3), (4), and (5). By doing so, much of the HTTP traffic going through the firewall will be off-loaded.

To take this scenario a step further, it may turn out that the caching proxy will become a bottleneck on the internal network. To alleviate this problem, several caching proxies can be setup within the internal network. By using a version of DNS that supports roundrobin queries (for example, bind v4.9.3), DNS client requests for the caching proxy's IP address will be given a different one from the available list of proxies. Effectively, this load balances the HTTP requests between all of the internal caching proxies.

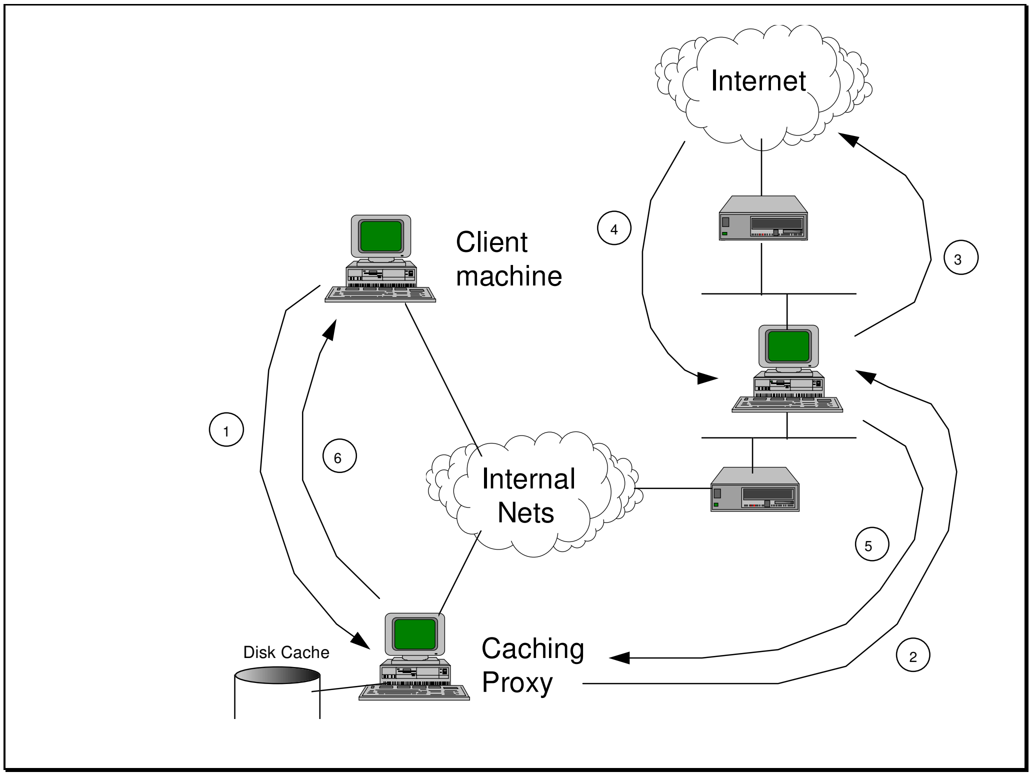

Figure 13. Caching HTTP Proxy Environment

GLOSSARY OF TERMS

BSD .... Berkeley Software Distribution

BSDI .... Berkeley Software Design, Inc.

CPU .... Central Processing Unit

DNS .... Domain Name System

FDDI .... Fiber Distributed Data Interface

FTP .... File Transfer Protocol

GB .... Gigabytes

GIF .... Graphics Image Format

HTML .... HyperText Markup Language

HTTP .... HyperText Transfer Protocol

IDENT .... Identification Protocol

IP .... Internet Protocol

Kbps .... Kilobits per second

LAN .... Local Area Network

MB .... Megabytes

Mbps .... Megabits per second

OS .... Operating System

TELNET .... Terminal Emulation over a Network

UUCP .... UNIX to UNIX Copy

URL .... Universal Resource Locator

WAN .... Wide Area Network

APPENDIX A — Test Scripts

TELNET Scripts

DEFIANT

```
#!/bin/sh
#
PATH=/bin:/usr/bin:/usr/ucb:/usr/5bin:/etc:/usr/etc; export PATH
Echo=/usr/5bin/echo

if [ $# -eq 0 ]; then
       $Echo "usage: $0 #_of_tests"
       exit
fi

#
#      fill DNS cache
#
nslookup relay.orgn.ashton.csc.com >/dev/null

N=$1

while [ "$N" -ge 1 ]
do
  echo Test $N
  (date>tn-$N; telnet relay.orgn</dev/console >/dev/null; date >>tn-$N)&
  N=`expr $N - 1`
done
```

BORG

```
#!/bin/sh
#
PATH=/bin:/usr/bin:/usr/ucb:/usr/5bin:/etc:/usr/etc; export PATH
Echo=/usr/5bin/echo

if [ $# -eq 0 ]; then
        $Echo "usage: $0 #_of_tests"
        exit
fi

#
#       fill DNS cache
#
nslookup relay.orgn.ashton.csc.com >/dev/null

N=$1
N=`expr $N + $N`
end=`expr $1 + 1`

while [ "$N" -ge "$end" ]
do
  echo Test $N
  (date>tn-$N; telnet relay.orgn </dev/console >/dev/null; date >>tn-$N)&
  N=`expr $N - 1`
done
```

Summary script

```
#!/usr/local/bin/perl
#
#      usage: telsumm.pl tn-* > file

$n_connects = $#ARGV + 1;
foreach (@ARGV) {
  $file = $_;
  $test_no = (split(/-/, $file))[1];

  open (IN, $file) || die "could not open file $file\n";
  $t1=<IN>; $t1 =~ tr/ / /s;
  $t2=<IN>; $t2 =~ tr/ / /s;
  close IN;

  $t1 = (split(/ /, $t1))[3];
  $t2 = (split(/ /, $t2))[3];

  $t1h= (split(/:/, $t1))[0];
  $t1m= (split(/:/, $t1))[1];
  $t1s= (split(/:/, $t1))[2];
  $time1 = $t1h*60*60 + $t1m*60 + $t1s;

  $t2h= (split(/:/, $t2))[0];
  $t2m= (split(/:/, $t2))[1];
  $t2s= (split(/:/, $t2))[2];
  $time2 = $t2h*60*60 + $t2m*60 + $t2s;

  $actual_time = $time2 - $time1;
  $mod_time = $actual_time - (4*60);

  $real[$test_no] = $actual_time;
  $setup[$test_no] = $mod_time;
}

for (1..$n_connects) {
  printf "Test # %3d      Actual Time: %d     Setup Time: %d\n", $_,
$real[$_], $setup[$_];
}
```

FTP Scripts

DEFIANT

```
#!/bin/sh
#
PATH=/bin:/usr/bin:/usr/etc:/usr/ucb:/etc; export PATH

Echo=/usr/5bin/echo

remfile=256K

$Echo "Testing $1 ftp connections..."

cat >/tmp/$$ <<END
get $remfile /dev/null
quit
END

nslookup relay.orgn.ashton.csc.com > /dev/null
nslookup voyager.ashton.csc.com > /dev/null
nslookup lightspeed.ashton.csc.com > /dev/null

N=$1

while [ "$N" -ge 1 ]
do
  $Echo "Test $N started..."
  (date>ftp-$N;ftp -iv voyager.ashton.csc.com </tmp/$$ >>ftp-$N;date>>ftp-$N)
&
  N=`expr $N - 1`
 (date>ftp-$N;ftp -iv lightspeed.ashton.csc.com </tmp/$$ >>ftp-$N;date>>ftp-
$N) &
  N=`expr $N - 1`
  sleep 1
done

rm -r /tmp/$$
```

BORG

```
#!/bin/sh
#
PATH=/bin:/usr/bin:/usr/etc:/usr/ucb:/etc; export PATH

Echo=/usr/5bin/echo

remfile=256K

$Echo "Testing $1 ftp connections..."

cat >/tmp/$$ <<END
get $remfile /dev/null
quit
END

nslookup relay.orgn.ashton.csc.com > /dev/null
nslookup voyager.ashton.csc.com > /dev/null
nslookup lightspeed.ashton.csc.com > /dev/null

N=$1
N=`expr $N + $N`
end=`expr $1 + 1`

while [ $N -ge $end ]
do
```

```
  $Echo "Test $N started..."
  (date>ftp-$N;ftp -iv voyager.ashton.csc.com </tmp/$$ >>ftp-$N;date>>ftp-$N)
&
  N=`expr $N - 1`
 (date>ftp-$N;ftp -iv lightspeed.ashton.csc.com </tmp/$$ >>ftp-$N;date>>ftp-
$N) &
  N=`expr $N - 1`
  sleep 1
done

rm -r /tmp/$$
```

Summary Script

```
#!/usr/local/bin/perl
#
#      usage: ftpsumm.pl ftp-* > file
@files = sort by_ext @ARGV;
foreach $f (@files) {
  open(IN, $f) || die "cannot open file $f\n";
  $line = "";
  @t = ();
  while (<IN>) {
     tr/ / /s;
     /EDT 1995/ && do { @t=(@t, (split(/ /, $_))[3]); };
     /bytes received/ && do { $line = $_; };
  }
  if (length($line) == 0) {
    die "recieve statistics not found in file $f\n";
  }
  close IN;
  ($bytes, $null1, $null2, $null3, $time, $rest) = split(/ /, $line);
  if (@t) {
     $t1 = $t[0];
     $t2 = $t[1];

     $t1h= (split(/:/, $t1))[0];
     $t1m= (split(/:/, $t1))[1];
     $t1s= (split(/:/, $t1))[2];
     $time1 = $t1h*60*60 + $t1m*60 + $t1s;

     $t2h= (split(/:/, $t2))[0];
     $t2m= (split(/:/, $t2))[1];
     $t2s= (split(/:/, $t2))[2];
     $time2 = $t2h*60*60 + $t2m*60 + $t2s;

     $time = $time2 - $time1;
  }
  $test_no = (split(/-/, $f))[1];
  printf "Test # %3d:  Time: %.3f      (%.5f Kbytes/s)\n", $test_no, $time,
($bytes/1024/$time);
  @time_list = (@time_list, $time);
}
$min = (sort by_num @time_list)[0];
$max = (sort by_num @time_list)[$#time_list];
foreach (@time_list) {
  $sum += $_;
}
$avg = $sum / $test_no;
print "\n";
printf "Minimum time: %.3f\n", $min;
printf "Average time: %.3f\n", $avg;
printf "Maximum time: %.3f\n", $max;


sub by_ext {
```

```
  (split(/-/,$a))[1] <=> (split(/-/,$b))[1];
}

sub by_num {
  $a <=> $b;
}
```

HTTP Scripts

DEFIANT - High Connect

```
#!/bin/sh
PATH=/usr/bin:/bin:/usr/local/bin:/usr/ucb:/etc:/usr/etc; export PATH
Echo=/usr/5bin/echo

nslookup relay.orgn.ashton.csc.com > /dev/null
nslookup voyager.ashton.csc.com > /dev/null
nslookup lightspeed.ashton.csc.com > /dev/null

doc=ascii_text.html

http_proxy=http://relay.orgn.ashton.csc.com:80/
export http_proxy

N=$1
while [ $N -gt 0 ]
do
  echo "Test $N..."
  (date>http-$N;lynx -source \
http://voyager.ashton.csc.com/$doc>/dev/null;date>>http-$N)&
  N=`expr $N - 1`
  (date>http-$N;lynx -source \
http://lightspeed.ashton.csc.com/$doc>/dev/null;date>>http-$N)&
  N=`expr $N - 1`
done
```

DEFIANT - High Volume

```
#!/bin/sh
PATH=/usr/bin:/bin:/usr/local/bin:/usr/ucb:/etc:/usr/etc; export PATH
Echo=/usr/5bin/echo

nslookup relay.orgn.ashton.csc.com > /dev/null
nslookup voyager.ashton.csc.com > /dev/null
nslookup lightspeed.ashton.csc.com > /dev/null

doc=1M       # specify file to get here

http_proxy=http://relay.orgn.ashton.csc.com:80/
export http_proxy

N=$1
while [ $N -gt 0 ]
do
  echo "Test $N..."
  (date>http-$N;lynx -source \
http://voyager.ashton.csc.com/$doc>/dev/null;date>>http-$N)&
  N=`expr $N - 1`
  (date>http-$N;lynx -source \
http://lightspeed.ashton.csc.com/$doc>/dev/null;date>>http-$N)&
  N=`expr $N - 1`
done
```

BORG - High Connect

```
#!/bin/sh
PATH=/usr/bin:/bin:/usr/local/bin:/usr/ucb:/etc:/usr/etc; export PATH
Echo=/usr/5bin/echo

nslookup relay.orgn.ashton.csc.com > /dev/null
```

```
nslookup voyager.ashton.csc.com > /dev/null
nslookup lightspeed.ashton.csc.com > /dev/null

doc=ascii_text

http_proxy=http://relay.orgn.ashton.csc.com:80/
export http_proxy

N=$1
N=`expr $N + $N`
end=`expr $1 + 1`

while [ "$N" -ge "$end" ]
do
  echo "Test $N..."
  (date>http-$N;lynx -source \
http://voyager.ashton.csc.com/$doc>/dev/null;date>>http-$N)&
  N=`expr $N - 1`
  (date>http-$N;lynx -source \
http://lightspeed.ashton.csc.com/$doc>/dev/null;date>>http-$N)&
  N=`expr $N - 1`
done
```

BORG - High Volume

```
#!/bin/sh
PATH=/usr/bin:/bin:/usr/local/bin:/usr/ucb:/etc:/usr/etc; export PATH
Echo=/usr/5bin/echo

nslookup relay.orgn.ashton.csc.com > /dev/null
nslookup voyager.ashton.csc.com > /dev/null
nslookup lightspeed.ashton.csc.com > /dev/null

doc=1M

http_proxy=http://relay.orgn.ashton.csc.com:80/
export http_proxy

N=$1
N=`expr $N + $N`
end=`expr $1 + 1`

while [ "$N" -ge "$end" ]
do
  echo "Test $N..."
  (date>http-$N;lynx -source \
http://voyager.ashton.csc.com/$doc>/dev/null;date>>http-$N)&
  N=`expr $N - 1`
  (date>http-$N;lynx -source \
http://lightspeed.ashton.csc.com/$doc>/dev/null;date>>http-$N)&
  N=`expr $N - 1`
done
```

Summary Script

```
#!/usr/local/bin/perl
#
#      usage: httpsumm.pl http-* > file

$n_connects = $#ARGV + 1;
foreach (@ARGV) {
  $file = $_;
  $test_no = (split(/-/, $file))[1];

  open (IN, $file) || die "could not open file $file\n";
  $t1=<IN>;
```

```
  $t2=<IN>;
  close IN;

  $t1 = (split(/ /, $t1))[3];
  $t2 = (split(/ /, $t2))[3];

  $t1h= (split(/:/, $t1))[0];
  $t1m= (split(/:/, $t1))[1];
  $t1s= (split(/:/, $t1))[2];
  $time1 = $t1h*60*60 + $t1m*60 + $t1s;

  $t2h= (split(/:/, $t2))[0];
  $t2m= (split(/:/, $t2))[1];
  $t2s= (split(/:/, $t2))[2];
  $time2 = $t2h*60*60 + $t2m*60 + $t2s;

  $actual_time = $time2 - $time1;

  $real[$test_no] = $actual_time;
}

for (1..$n_connects) {
  printf "Test # %3d      Time: %d\n", $_, $real[$_];
}
```

Email Scripts

Used by both LIGHTSPEED and ENTERPRISE

```
#!/bin/sh
#
PATH=/bin:/usr/bin:/etc:/usr/etc:/usr/ucb; export PATH
Echo=/bin/echo

if [ $# -eq 0 ]; then
        $Echo "usage: $0 #_of_tests"
        exit
fi

doc=1.2m.uu

nslookup relay.orgn.ashton.csc.com >/dev/null

N=$1
while [ $N -ge 1 ]
do
  echo Test $N
  Mail -s "Test $N (`hostname`)" user1@orgn.ashton.csc.com < $doc
  Mail -s "Test $N (`hostname`)" user2@orgn.ashton.csc.com < $doc
  Mail -s "Test $N (`hostname`)" user3@orgn.ashton.csc.com < $doc
  Mail -s "Test $N (`hostname`)" user4@orgn.ashton.csc.com < $doc
  N=`expr $N - 4`
done
```

Netperm-table

Script used to create the rule sets

```
#!/usr/bin/perl
#
if (! @ARGV) {
  die "usage: $0 no_of_rules\n";
}

@addrs = ("195.1.1", "195.1.2", "195.1.3", "195.1.4", "195.1.5", "195.1.6",
"195.1.7", "195.1.8", "196.1.1", "196.1.2", "196.1.3", "196.1.4", "196.1.5",
"196.1.6", "196.1.7", "196.1.8", "197.1.1", "197.1.2", "197.1.3", "197.1.4",
"197.1.5", "197.1.6", "197.1.7", "197.1.8", "198.1.1", "198.1.2", "198.1.3",
"198.1.4", "198.1.5", "198.1.6", "198.1.7", "198.1.8");

system ("cat netperm-table");

$c = 0;
LOOP:
foreach (@addrs) {
  foreach $h (2..254) {
    print "tn-gw:   deny-hosts ${_}.$h\n";

    if ($c++ > $ARGV[0]) {
      last LOOP;
    }
  }
}
exit 0;
```