

Zeus Technology

Web Server Performance and Scalability

Date: 21/11/00
Author: Neil Macehiter
Version: 1.1

Zeus Technology
Zeus House
Cowley Road
Cambridge
CB4 0ZT
UK
Phone: +44 1223 525000
Fax: +44 1223 525100
E-mail: info@zeus.com
Web: <http://www.zeus.com/>

Zeus Technology
Suite 340
5201 Great America Parkway
Santa Clara
CA 95054
USA
Phone: +1 408 350 9400
Fax: +1 408 350 9408

Copyright © 2000 Zeus Technology.

All rights reserved. No part of this document may be reproduced in any form, including photocopying or transmission electronically to any computer, without prior written consent of Zeus Technology. The information contained in this document is confidential and proprietary to Zeus Technology, and may not be used or disclosed except as expressly authorized in writing by Zeus Technology.

Table of Contents

1	Executive Summary	3
2	Zeus Web Server: Designed For Scalability.....	5
3	Benchmark Results	6
3.1	Scenario One: Intranet Web Serving.....	6
3.2	Scenario Two: Dial-Up/Mobile Web Serving.....	8
3.2.1	Interpretation	10
3.2.2	Why Is Apache's Performance So Poor?.....	11
3.3	Scenario Three: Broadband Web Serving	11
3.4	Scenario Four: Processor Scalability	12
4	Conclusions	13
5	About Zeus	14
	Appendix A ApacheBench.....	15
	System Details.....	15
	The Tests.....	15
	Networking Setup	15
	Static Content	15
	Web Server Configuration Files	16
	Appendix B Simulating Dial-up/mobile Connections	18
	Class-Based Queuing.....	18
	Implementation and Deployment.....	19
	Performance Issues.....	20
	The Tests.....	21
	Test Overview.....	21
	Test Tools.....	21
	Methodology.....	22
	ratelimit	22
	Zeus Web Server Configuration	27
	Apache Web Server Configuration.....	29
	Router Kernel Configuration	32

1 Executive Summary

The Internet is irreversibly changing the way that we all live. The way we communicate, the way we work and the way we play. Whether you are an ISP providing hosting services, an internet content provider, or an enterprise operating in B2C, B2B or B2E markets, the Internet presents massive opportunities. Exploiting those opportunities presents significant challenges:

- Moore's law predicts that computers will become twice as fast and twice as cheap every 18 months (59% growth per annum). The number of users online is growing at 91% per annum¹ and the amount of data traffic on the world's largest networks has been growing by 1000% consistently for the last three years^{2 3 4}
- According to Zona Research an average surfer will leave your site after only 8 seconds if their request is not served with the result that perhaps as much as \$4.35 billion in ecommerce sales in the U.S. may be lost each year.⁵
- eBay lost \$5M in fees and 20% of their market capitalization whilst traffic to their competitor sites increased by 50% as a result of the inability to satisfy user demand.⁶
- Contingency Research Planning estimate that failing to meet service level agreements costs a retail brokerage \$6.5M/hour.⁷
- "If we don't perform well, we risk something much more important than dollars: our customers' goodwill" (Motley Fool).⁸

To successfully meet these challenges requires the deployment of Web infrastructure with the following capabilities:

- Predictable, cost-effective scalability
- Performance: response time and throughput
- 7*24 availability
- Application integration
- Comprehensive security
- Enterprise class administration and control
- Vendor-supplied integration and support services

The web server is a critical component of the web infrastructure: it manages the dialogue between the thousands of concurrent clients and your backend systems. This

¹ NUA Internet Surveys "How many Online?" http://www.nua.ie/surveys/how_many_online/world.html

² John Sidgmore, CEO UUNET, Network + Interop in Las Vegas May 1998, http://www.internetnews.com/bus-news/article/0,,3_20531,00.html

³ Vinton Cerf, senior vice president of Advanced Internet Technology and Architecture for MCI. <http://www.alcatel.com/telecom/mbd/publi/newslink/9902/interview.htm>

⁴ John Sidgmore, CEO UUNET, New York Infotech Forum, January 2000 http://www.insight-corp.com/newsfirst/v8_ie1.html

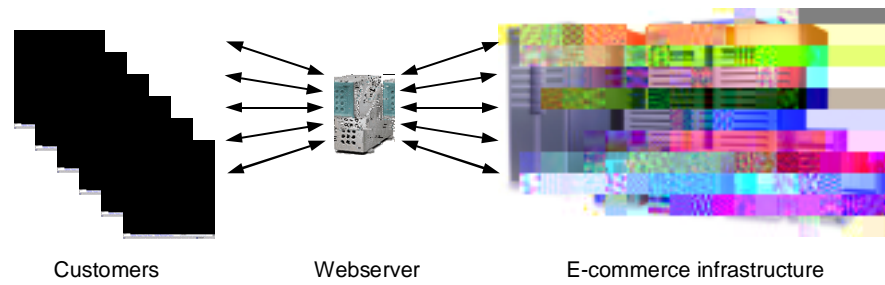
⁵ Zona Research, The Economic Impacts of Unacceptable Web Site Download Speeds, http://www.zonaresearch.com/deliverables/white_papers/wp17/

⁶ Industry Standard, 06 Sept 2000, Flirting with Disaster <http://thestandard.com.au/articles/display/0,1449,6176,00.html>

⁷ Industry Standard, 06 Sept 2000, Flirting with Disaster <http://thestandard.com.au/articles/display/0,1449,6176,00.html>

⁸ Internet Week Online, The Cost Of Downtime <http://www.internetwk.com/lead/lead073099.htm>

concentration effect means that the web server plays a pivotal role in providing the customer experience.



This paper focuses on the performance and scalability characteristics of the Zeus Web Server. It demonstrates that Zeus Web Server enables the deployment of mission-critical web infrastructure in terms of:

- Rapid end-user response time
- Transaction throughput
- Scalability in terms of the number of connections for a given quality of service
- Predictable performance with increased workload and available resources
- Maximising return on hardware investment
- Ability to exploit multiple processors

2 Zeus Web Server: Designed For Scalability

The Zeus Web Server is the most scalable web server software on the market. It has been engineered to host the world's most demanding web sites. The key difference between web server architectures is how they handle multiple simultaneous connections. This difference directly affects both the fundamental performance and scalability of the solution.

The unique Zeus Web Server architecture utilizes a small, fixed number of single-threaded I/O multiplexing processes, each capable of handling tens of thousands of simultaneous connections, coupled with a kernel-threaded component for high-performance dynamic content.

As the traffic to your web site grows, a single machine, no matter how large, will reach a point where it is unable to provide the levels of availability and performance you require. Clustering will become increasingly common given the fact that the improvements in hardware performance predicted by Moore's law are not keeping pace with the growth in user numbers and traffic.

The Zeus Web Server has been designed with clustering capabilities from the outset. It has unique provisioning and management capabilities, which make managing clustered web server farms quicker and easier.

3 Benchmark Results

Web servers are deployed in a broad spectrum of environments, on hardware ranging from commodity 1U appliance servers to the latest multiprocessor mini computers and networks with bandwidths in the tens of kilobits to multiple gigabits per second. In order to demonstrate the performance and scalability benefits of the Zeus Web Server in these environments, the results of a series of benchmark tests are presented. Each test has been designed with different objectives:

- To test web server throughput and performance under conditions representative of Intranet deployments using appliance hardware
- To test end-user response time, web server throughput and scalability in real-world conditions of dial-up modem/slow network connectivity
- To test the performance and scalability of the Web server without hardware constraints and utilizing next generation networks
- To test how web server throughput scales with additional CPU resource

Scenarios one and two provide a comparison of Zeus Web Server and Apache. Apache was used because the licence agreement for iPlanet Enterprise Server prevents the publication of benchmark results and Microsoft's Internet Information Services is not available on UNIX/Linux.

3.1 Scenario One: Intranet Web Serving

The Apache Foundation's ApacheBench benchmark was used to test the performance of Zeus Web Server and Apache in a typical Intranet deployment on commodity appliance hardware⁹. For further details of the benchmark configuration and test, refer to Appendix A.

ApacheBench was used to make simultaneous requests to a web server for a single 1000 byte file. The number of simultaneous requests was varied in order to understand how server throughput varied with the requested load as shown in Figure 1. Page size (content length) was then varied with the results shown in Figure 2.

Figure 1 shows that Apache is limited to approximately 1500 completed requests per second while Zeus Web Server scales to more than four times this figure when it becomes limited by network bandwidth. Figure 2 shows that across the range of content lengths, subject to network bandwidth, Zeus Web Server consistently provides a 200%-300% throughput advantage.

⁹ Intel ISP 1100 with Celeron 500 MHz, 128MB of RAM, 4GB HDD and Intel EtherExpress Pro 100 network card (100 Mbit/s), running RedHat 6.0 Linux

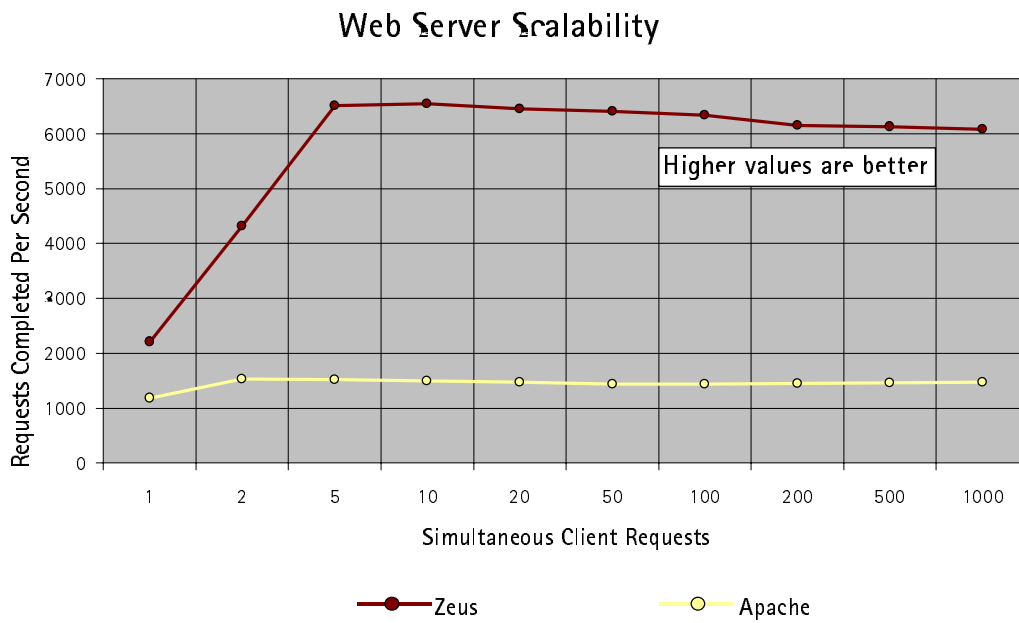


Figure 1 Web Server Scalability

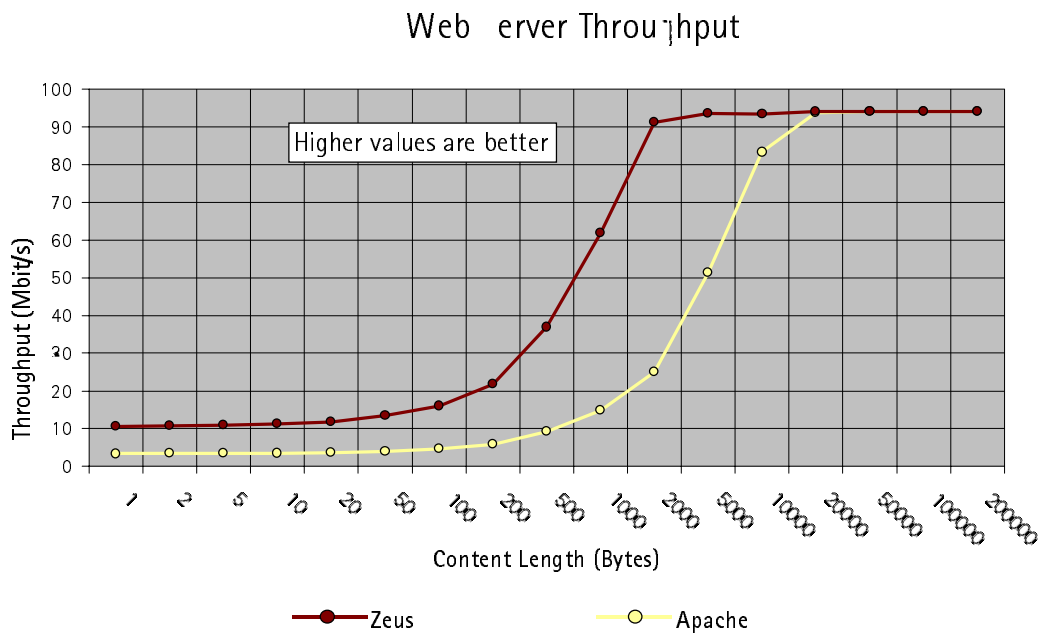


Figure 2 Web Server Throughput

3.2 Scenario Two: Dial-Up/Mobile Web Serving

Web server benchmarks are often carried out over fast networks, with low latency and high bandwidth. In real-world scenarios, many of the visitors to a web site will be using slow dial-up/mobile modem connections, or will be connected to the web server across a number of slow networks. Each of these connections might easily be sustained for several seconds, and the web server must maintain some internal resources - perhaps memory, processes, or threads - for the duration of each connection.

This series of tests considers how web servers handle this pattern of load. The test was based on a Hewlett-Packard N-class as the system under test, with 4 CPUs, 8 GB of memory, and a 1 Gbit/s network interface card. To simulate dial-up/mobile connections, the rate of flow from each client was limited to 28,800bps using router traffic control features. For further details of the benchmark configuration and test, refer to Appendix B.

Figure 3 shows the effect of simultaneous connections on server throughput. Figure 4 shows the effect of the number of connections per second on the average time taken between the client sending the first byte of the request and receiving the first byte of the reply. This test is of significance given the Zona Research analysis, which indicates that customers will leave a site if their results of their request are not downloaded in less than 8 seconds. Figure 5 shows the effect of the number of connections per second on the average total time taken by the server to transfer the entire reply to the client.

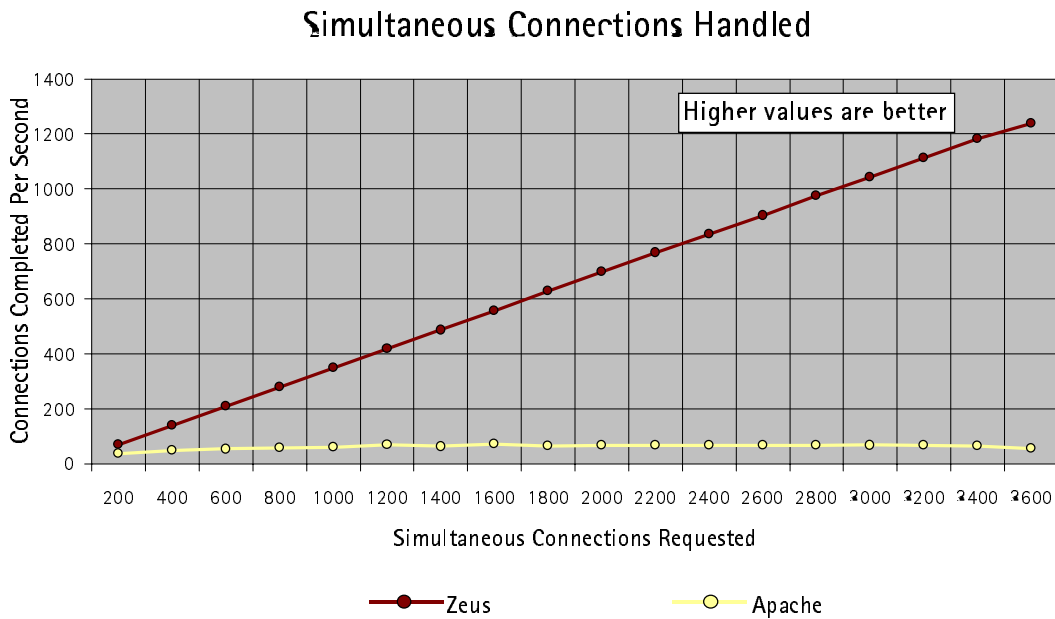


Figure 3 Simultaneous Connections Handled

Initial Response Times

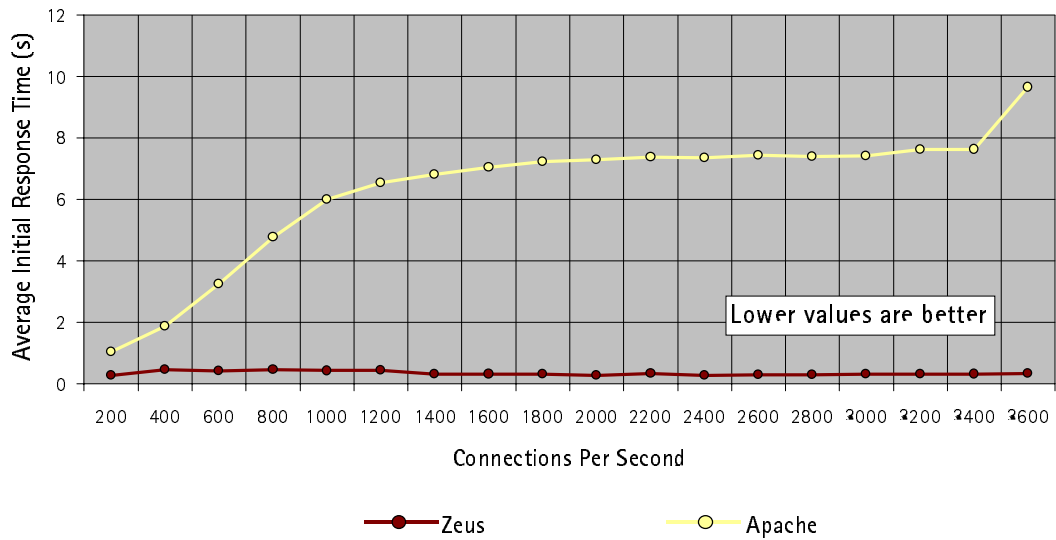


Figure 4 Initial Response Times

Transfer Times

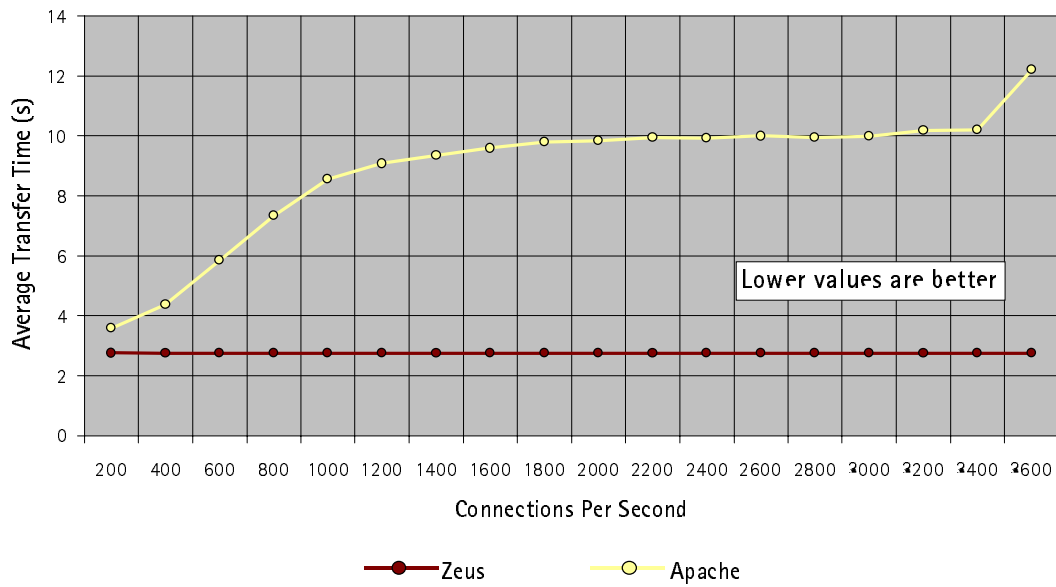


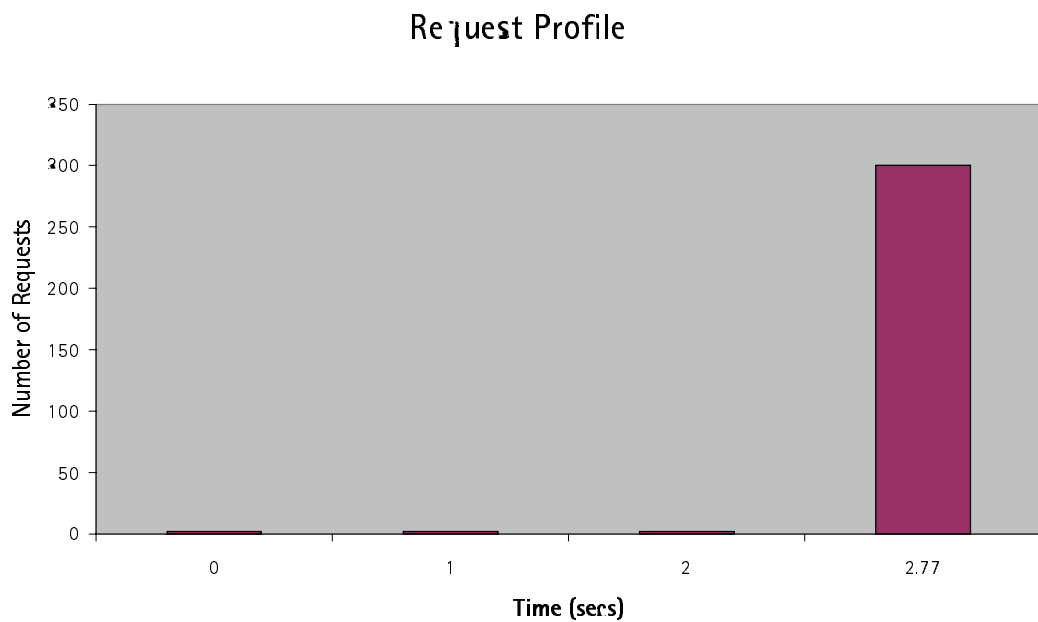
Figure 5 Transfer Times

3.2.1 Interpretation

For the test in Figure 3 the minimum time to transfer a 10000 byte file over a 28800 bits/sec modem is 2.77 seconds, based on the following formula:

$$\text{Transfer Time (sec)} = \frac{10000 * 8 \text{ bits}}{28800 \text{ bits/sec}}$$

Consider the situation where 300 simultaneous requests are made at time 0. The following graph illustrates the number of requests completed over time from time 0:



The theoretical average requests completed per second is 108 (300/2.77).

The maximum number of connections which can be completed per second (y) based on the number of simultaneous connections (x) is thus governed by the following formula:

$$y = x * \frac{\text{Baud rate (bits/sec)}}{\text{File size (bits)}}$$

For the test in 3.2.1,

$$y = x * \frac{28800 \text{ bits/sec}}{80000 \text{ bits}}$$

i.e. $y = x * 0.36$

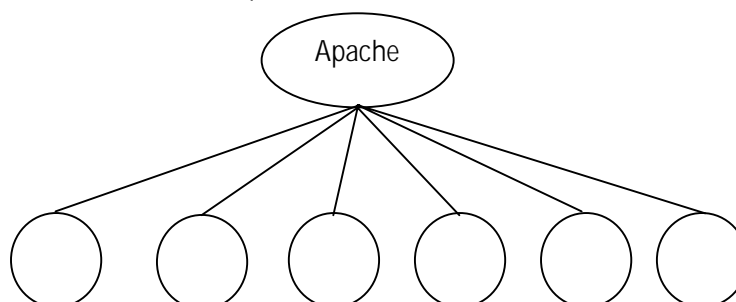
An examination of the test results and graph for Zeus Web Server in 3.2.1, illustrates that Zeus Web Server is capable of sustaining simultaneous connections at close to the theoretical maximum.

Simultaneous Connections Requested	Connections completed/sec by Zeus Web Server	Connections completed/sec by Apache	Theoretical connections completed/second
200	69	37	72
1000	349	60	360
2000	697	66	720
3600	1239	55	1296

The tests in Figure 4 and Figure 5 illustrate that Zeus Web Server delivers significantly better response time than Apache and that significantly larger numbers of connections/second can be supported for the same quality of service. At 1000 connections/second the Apache server throughput has degraded to the point where users (based on the 8 second rule) will leave a site. Taking into account the fact that the tests were performed with a simple 10,000 byte file and that typical Web pages include graphics, images each of which require a separate request, it is clear that Zeus Web Server is capable of delivering the response times required in real-world scenarios, in contrast to Apache where the response time rapidly degrades to 8 seconds.

3.2.2 Why Is Apache's Performance So Poor?

The architecture of the Apache web server is based on the creation of separate processes for each connection request:



There is a hard limit of 256 for the number of these processes which can be created (a limit which can only be altered by changing and recompiling the Apache source code). Given the formula above, this means that the maximum number of connections which can be completed per second by Apache is 92 (256*0.36). This limit explains the flat graph for Apache in 3.2.1 and the figures provided in the table above.

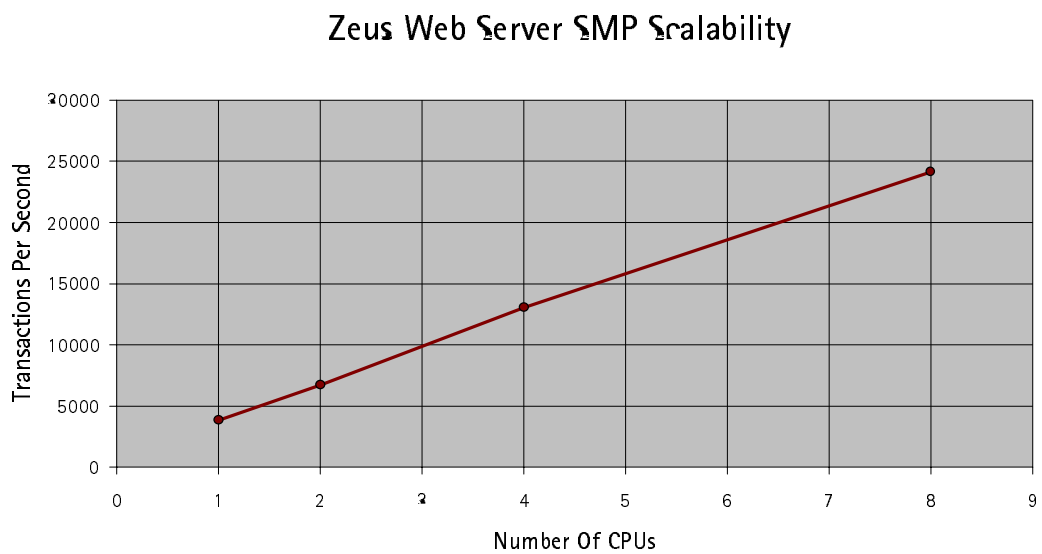
3.3 Scenario Three: Broadband Web Serving

This decade will see the global deployment of broadband access networks, including cable, xDSL and wireless. Typically these technologies will provide users with upwards of 250 kbps Internet connectivity. Recognising this, in 1999 SPEC introduced its second-generation web server benchmark (refer to <http://www.specbench.org/osg/web99/> for further information), which measures the number of simultaneous broadband connections that a server can sustain. The SPECweb99 benchmark includes many sophisticated and state-of-the-art enhancements to meet the modern demands of web users of today and tomorrow, such as dynamically generated content.

On the 4th October 2000, IBM and Zeus achieved a new world record for web server performance. The SPECweb99 result of 7288 is equivalent to serving data at a sustained rate of 2.6Gbps - this is first SPECweb99 result equivalent to saturating an OC-48 (2.488Gbps) pipe (refer to <http://www.zeus.com/news/articles/001004-001/> for more information).

3.4 Scenario Four: Processor Scalability

In the following published SPECweb96 benchmarks using a Hewlett Packard 9000 N4000 class machine, the number of transactions/second supported by the Zeus Web Server was measured as the number of CPUs was increased:



Zeus Web Server exploits symmetric multiprocessing platforms with near linear scaling with additional CPU resource.

4 Conclusions

The web server is a critical component of your web infrastructure. It manages the dialogue between the thousands of concurrent clients and your backend systems. This concentration effect means that the web server plays a pivotal role in providing the customer experience.

The performance and scalability of your web server is critical to delivering the highest service levels to your customers; deploying an architecture which can grow with your business; to maximise the return on investment in servers; and to exploit low-cost Internet server appliances.

The results of the benchmarks demonstrate that Zeus Web Server meets these critical requirements, providing:

- Four times the number of simultaneous connections of Apache
- One third of the response time and three times the throughput of Apache
- Consistent quality of service with increasing workload
- Performance which scales with workload under real-world conditions
- Near linear scaling of throughput with processors on symmetric multiprocessing platforms

Conclusions which are verified by the experience of Zeus customers using Zeus Web Server:

- "Zeus provides a solid foundation to the services we offer, complimenting the high performance, scalability and support that our clients have come to expect." (Virtual Internet)
- "Zeus is the fastest web server I have ever encountered ... we save more on machines than the software costs ... it's serving over a million images a day and generating barely noticeable overhead" (Go2Net)
- "We use Zeus Webserver v3 for our production websites. Zeus is an amazing product, that far out performs any freeware webserver available. Zeus' ultra high performance allows us to fully leverage our hardware investments, while it's unique clustering feature reduces administration time by over 200%." (CAT Internet <http://www.catinternet.com/ourpartners.html>)
- "Historically we've always used Apache servers ... In November 1999 we changed to Zeus for our business web hosting and one of our main corporate websites ... Our performance is both fast and reliable, and we're even able to use our main commercial service hardware for other tasks as the load on the operating system caused by the webserver is low and easily manageable." (Telewest's ISP blueyonder)

5 About Zeus

Zeus Technology develops the world's leading web serving technologies, designed to provide the fundamental building blocks of a scalable Internet infrastructure.

Best known for our flagship product, Zeus Web Server, we provide a broad range of products to enable the new generation of web-based computing. If you have used the Internet, it is more than likely that you've visited one of the significant number of web sites that are powered by the Zeus Web Server.

Zeus software currently powers 1 in 35 of the world's web sites. Our competitive advantage derives from the speed of the software and tremendous scalability. Zeus Web Server consistently dominates performance benchmarks.

As testament to the technical superiority of our products, we have built up an impressive global customer base. Our customers include eBay, NetZero, Cable and Wireless, NTL, BT, Telefonica, Lycos, Infoseek, OpenIPO.com, Blue Mountain and UUNet.

Appendix A ApacheBench

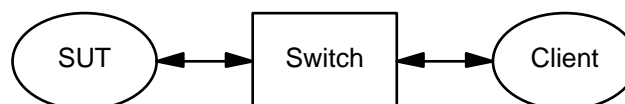
System Details

	Zeus	Apache
<i>Number of CPUs</i>	1	1
<i>CPU Type</i>	Intel Celeron	Intel Celeron
<i>CPU Speed</i>	500 MHz	500 MHz
<i>Memory</i>	128 MB	128 MB
<i>Disk Subsystem</i>	IDE	IDE
<i>Hard Disk</i>	4.8 GB	4.8 GB
<i>Number of NICs</i>	2	2
<i>Interface Types</i>	10/100Mbps Ethernet	10/100Mbps Ethernet
<i>Operating System</i>	Linux 2.2.15 (Redhat 6.0 with latest 2.2.x kernel)	Linux 2.2.15 (Redhat 6.0 with latest 2.2.x kernel)
<i>Web Server Version</i>	3.3.7	1.3.12
<i>Perl version</i>	5.6.0	5.6.0

The Tests

Networking Setup

The system under test (SUT) was connected to a single client machine via a 3COM SuperStack II Switch 3300. The same client machine was used for all tests. The network diagram is shown below:



The switch was configured to allow 100Mbit full-duplex connections between the SUT and the test client. The test client was chosen to be faster than the SUT. During all tests, it was ensured that the client machine was never the limiting factor in the benchmark.

Static Content

ApacheBench was used to test static content throughput of the web server. This tests network throughput and raw web serving performance. The test was carried out over a range of content lengths, with results shown below. ApacheBench was invoked with:

- 20 concurrent connections
- Run time proportional to log(filesize)
- HTTP keepalive enabled

Web Server Configuration Files

The global.cfg file:

```
controlport 9080
controlallow 127.0.0.1
errlog %zeushome%/web/log/errors
gid daemon
uid daemon
```

Virtual server configuration file:

```
dnslookup yes
docroot /space/webpages
env!inherited!PATH
env!inherited!TZ
ip_name amd1
modules!access!enabled no
modules!access!ldap!enabled no
modules!cgi!allowanywhere yes
modules!cgi!allowcmd yes
modules!cgi!autoid no
modules!cgi!chroot no
modules!cgi!enabled yes
modules!cgi!ulimitas 0
modules!cgi!ulimitcpu 0
modules!cgi!ulimitdata 0
modules!cgi!ulimitnproc 0
modules!dirlist!enabled yes
modules!distributed!enabled no
modules!distributed!servlet!prefix /servlet
modules!distributed!sockd!prefix /distributed
modules!errors!directory %zeushome%/web/etc/errpages
modules!errors!enabled yes
modules!fastcgi!enabled yes
modules!frontpage!enabled no
modules!frontpage!fphome /usr/local/frontpage
modules!get!enabled yes
modules!htaccess!accessfilename .htaccess
modules!htaccess!enabled no
modules!imagemap!enabled yes
modules!index!enabled yes
modules!index!files index.html, index.htm, index.shtml,
index.cgi
modules!isapi!enabled no
modules!jserv!enabled no
modules!log!enabled no
modules!log!format %h %l %u %t "%r" %s %b
modules!map!alias!/cgi-bin!filepath /space/webpages/cgi-bin
modules!map!alias!/cgi-bin!type fastcgi
modules!map!alias!/icons!/filepath %zeushome%/web/etc/icons/
modules!map!alias!/icons!/type simple
modules!map!alias_types!cgi application/x-httpd-cgi
modules!map!alias_types!fastcgi application/x-httpd-fcgi
modules!map!alias_types!isapi application/x-httpd-isapi
modules!map!alias_types!simple application/x-httpd-alias
modules!map!enabled yes
modules!map!homedir!dir public_html
modules!map!homedir!enabled yes
modules!map!homedir!file /etc/passwd
modules!map!homedir!ldap_filter cn=%u
modules!mime!default text/plain
modules!mime!enabled yes
modules!mime!types!Z application/octet-stream
modules!mime!types!api application/x-httpd-isapi
modules!mime!types!asis httpd/send-as-is
modules!mime!types!cgi application/x-httpd-cgi
modules!mime!types!exe application/octet-stream
```

```
modules!mime!types!gif image/gif
modules!mime!types!gz application/octet-stream
modules!mime!types!htm text/html
modules!mime!types!html text/html
modules!mime!types!jpeg image/jpeg
modules!mime!types!jpg image/jpeg
modules!mime!types!map application/x-httpd-imap
modules!mime!types!mov video/quicktime
modules!mime!types!mpeg video/mpeg
modules!mime!types!pdf application/pdf
modules!mime!types!shtml text/x-server-parsed-html
modules!mime!types!tar application/octet-stream
modules!mime!types!tgz application/octet-stream
modules!mime!types!txt text/plain
modules!mime!types!vrml x-world/x-vrml
modules!mime!types!wbmp image/vnd.wap.wbmp
modules!mime!types!wml text/vnd.wap.wml
modules!mime!types!wmlc application/vnd.wap.wmlc
modules!mime!types!wmls text/vnd.wap.wmlscript
modules!mime!types!wmlsc application/vnd.wap.wmlscriptc
modules!mime!types!wrl x-world/x-vrml
modules!nsapi!config_root %zeushome%/ns-config
modules!nsapi!enabled no
modules!put!enabled no
modules!put!save_as_user no
modules!search!enabled no
modules!search!input_template
%zeushome%/web/etc/search_form.html
modules!search!output_template
%zeushome%/web/etc/search_output.html
modules!search!url /search
modules!spelling!enabled yes
modules!stats!enabled yes
modules!subserver!enabled no
modules!throttle!enabled no
modules!throttle!limit 128000
modules!throttle!maxconns 50
modules!throttle!subserver no
port 80
security!client_cert 0
security!enabled no
```

Appendix B Simulating Dial-up/mobile Connections

Class-Based Queuing

Class-based queuing (CBQ) is the model used to simulate multiple slow connections. This section describes the general theory and provides more detail on the technology.

CBQ was designed to enable policy specifications to be combined with specified quality of service (QoS) guarantees. It causes traffic to obey a hierarchy of constraints. For example, if two organizations fund a backbone network link they might want bandwidth on that link to be allocated in proportion to their spending, and each organization might then wish to impose similar constraints upon their customers. These constraints are enforced by several major components. The terminology varies, but here we will use that used in the Linux implementation of CBQ for these components.

- *Queueing disciplines* are associated with each network device, as well as some more fine-grained entities. The simplest queueing discipline would just be a first-in-first-out queue, sending packets as quickly as possible in the order in which they were enqueued. Other queueing disciplines impose limits on the rate of flow of packets, or change the order in which packets are removed from their queues.
- *Classes* of packets are distinguished according to policy constraints. Each of them may be processed in a variety of ways depending on the queueing discipline in force, for instance by sending packets in order of the priority of their class. Since the class objects often use a further queueing discipline for their packets, a constraint hierarchy can be constructed as discussed above.
- *Filters* analyse properties of packets and assign them to appropriate classes.

Simulating a single dial-up/mobile link is thus a matter of choosing a queueing discipline that models such a link. For this, we have used a "token bucket filter", which constrains packets to be transmitted at no more than a given bit rate over any time interval. While this does not model random effects of slow network links, it does model the necessary bandwidth constraints.

Simulating a large number of dial-up/mobile links is conceptually simple, but requires a little care to avoid imposing too much load upon the router. One possible design consists of an array of classes, each of which has attached to it a filter catching packets addressed to a single port and a queueing discipline describing the limits to be imposed on traffic to that port.

Unfortunately, the Linux CBQ implementation has some difficulty with very large numbers of classes, which causes a noticeable degradation in router performance under heavy load. To achieve greater efficiency, we notice that filters can perform simple traffic policing using a token bucket filter, and dispense with the arrays of classes and queueing disciplines.

To keep the system under test (SUT) and the clients as close to the real world as possible, we must avoid extra load imposed by managing network traffic. No traffic control takes place on either the SUT or the clients; intermediate systems are used as routers instead.

User-level documentation on the Linux implementation of CBQ is rather fragmented. For instance, at the time of writing there is no user manual available with the `tc` utility, although some good papers and "how-to"-style projects are available elsewhere.

This whitepaper includes a utility called `ratelimit` which, although much less general than `tc`, should be easier to use for the purposes of the tests described here.

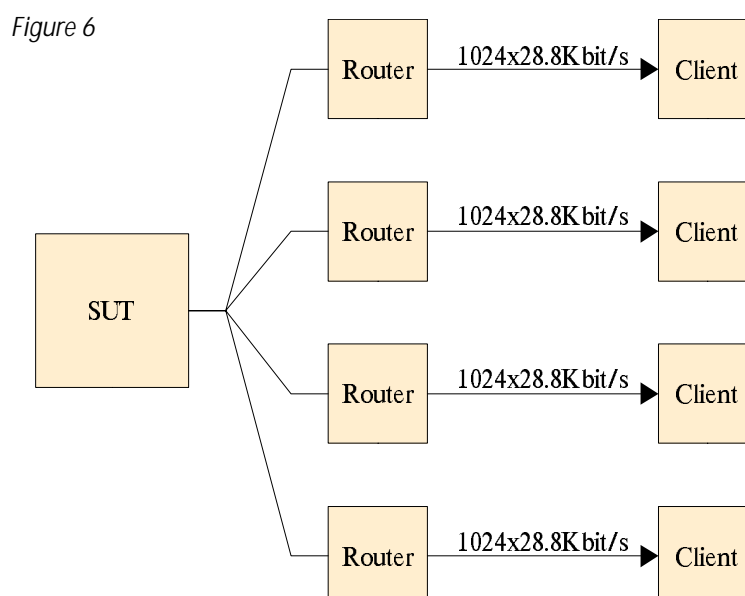
Implementation and Deployment

This section describes how the technology described above was deployed to simulate dial-up/mobile connections.

An implementation issue with the design described in the previous section is that the Linux CBQ code has no easily accessible concept of a connection, unless other tools are also used. To keep the implementation as simple as possible, we merely impose limits on (host, port) combinations; assuming that there is only one system under test, such combinations uniquely specify a connection. By the time the connection can be reused, the token bucket filter will already have returned to its initial state.

To avoid having to add a filter for each possible TCP port, it is useful to know in advance which client ports will be used by the benchmarking tools in question. Often this will consist of the "ephemeral ports", 1024 to 4999, which are used by connections not bound to specific client ports. If the range available for such ports is decreased, less load will be imposed upon the router; on Linux, the `/proc/sys/net/ipv4/ip_local_port_range` sysctl can be used to control this.

With these constraints in mind, the program `ratelimit` below may be used to set up rate-limiting on each of any number of routers. Consider, for instance, the network architecture shown in Figure 1:



Once the necessary network connections and routing tables have been set up, this architecture can have rate-limiting applied to it by using a single command similar to the following on each router:

```
ratelimit --interface eth0 --rate 28800 --hosts client \
--ports 1024-2047 --direction dest
```

This limits the rate of flow of all network traffic from the router to its client to 28.8 kilobits per second. Traffic from client to router is not limited, but '--direction

both' will rectify this.

To inspect the current state of the kernel's packet queueing structures, commands like the following may be used:

```
ratelimit --interface eth0 --list
ratelimit --interface eth0 --statistics
```

Performance Issues

It was observed that streamlining the CBQ structures in the Linux router yielded significant improvements in benchmark performance. The natural conclusion is that the router may become a bottleneck in simulations of this type. Therefore, in order to achieve maximum performance, there should be exactly one client per router, each with a number of rate-limited connections.

On a 100Mbit/s connection, the theoretical limit on the number of simulated 28.8Kbit/s connections is 3472, assuming there are no overheads. However, in practice TCP overhead and the dropped packets caused by imposing rate-limiting will reduce this limit; it is realistic to expect to be able to simulate about 1000 28.8Kbit/s modems on a 100Mbit/s network interface card (NIC).

Client performance may also become an issue under high loads. For instance, TCP/IP stacks must wait a certain period - often two minutes - before they reuse client ports, in order that packets are not mistakenly identified as belonging to previous connections. Since the available range of TCP port numbers is from 0 to 65535, a TCP/IP stack cannot sustain the creation of more than about 500 connections per second to a single port on a single remote host. In addition, clients that do not bind to an explicit client port will be assigned to one of the ephemeral ports; since these range from 1024 to 4999 in the default configuration on many systems, including Linux, that range can only sustain about 33 such connections per second. This is generally not high enough to place servers under stress, but using too many ports can degrade router performance; in this case, the number of clients should be increased.

The Tests

Test Overview

In order to examine the effects of large numbers of slow connections on web servers, several tests were carried out using a Hewlett-Packard N-class as the system under test (SUT). This system has 4 CPUs, 8 GB of memory, and a 1 Gbit/s NIC. The comparison was between Zeus Web Server 3.3.7 (see below) and Apache 1.3.12 (see below).

The system under test was connected to four routers via a single 3COM SuperStack II Switch 3300. Each router was running Linux 2.2.15 on an Intel Celeron 500, with its kernel configured as shown below, and was connected to a single client via the same switch. The clients were identical to the routers except that they had no special kernel configuration. All of the clients and the routers were connected using 100 Mbit/s NICs. This configuration is that shown in Figure 1 above.

Each router was configured to limit the rate of flow of network traffic from the SUT to its client to 28.8 Kbit/s on ports 1024 to 2047. A copy of `httperf` was then run on each client, and a number of simultaneous connections was attempted from each, ranging from 50 to 900. In order to keep the content served from each web server as simple as possible, all connections were to retrieve the same 10,000 byte file.

The results below were obtained by using four clients and four routers. Removing one client and one router did not affect the results; this implies that neither the routers nor the clients were a limiting factor in the benchmarks.

Test Tools

`httperf` is a tool to measure web server performance, offering a variety of means of generating workload. It tries to gain maximum utilization from the client machine, and computes all the results discussed here by default.

Methodology

Each time `httperf` was run, it was started simultaneously on each of the four clients, synchronizing via a separate TCP socket. The arguments to `httperf` were as follows, with the hostname of the system under test substituted for ``sut'` and the number of connections substituted for ``connections'`:

```
httperf --server sut --port 80 --uri /size/size10000 \  
--wsess connections,20,0 --burst-length 1 --timeout 6000
```

With these arguments, the output from an `httperf` run looked like the following:

```
Maximum connect burst length: 2

Total: connections 500 requests 10000 replies 10000 test-duration  
57.433 s

Connection rate: 8.7 conn/s (114.9 ms/conn, <=500 concurrent  
connections)  
Connection time [ms]: min 54407.3 avg 55046.0 max 56496.8 median  
55030.5 stddev 472.8  
Connection time [ms]: connect 19.8  
Connection length [replies/conn]: 20.000

Request rate: 174.1 req/s (5.7 ms/req)  
Request size [B]: 77.0

Reply rate [replies/s]: min 125.6 avg 174.4 max 274.4 stddev 40.6 (11  
samples)  
Reply time [ms]: response 262.3 transfer 2487.2  
Reply size [B]: header 169.0 content 10000.0 footer 0.0 (total  
10169.0)  
Reply status: 1xx=0 2xx=10000 3xx=0 4xx=0 5xx=0

CPU time [s]: user 2.45 system 54.98 (user 4.3% system 95.7% total  
100.0%)  
Net I/O: 1742.2 KB/s (14.3*10^6 bps)

Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0  
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

Session rate [sess/s]: min 0.00 avg 8.71 max 19.20 stddev 5.79  
(500/500)  
Session: avg 1.00 connections/session  
Session lifetime [s]: 55.0  
Session failtime [s]: 0.0  
Session length histogram: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 500
```

Of these, the number of simultaneous connections handled per second is taken from the average reply rate, the average response time is taken from the ``response'` part of the reply time, and the average transfer time is taken from the ``transfer'` part of the reply time. Each of these is summed over all clients.

ratelimit

The following Perl script was used to set up rate-limiting on each router in the tests:

```
#!/usr/bin/perl -w  
use strict;  
  
# ratelimit v0.3, copyright (c) 2000 Zeus Technology Limited.  
  
# Requires Linux 2.1.99 or greater with at least the following kernel
```

```

# configuration options:
# CONFIG_EXPERIMENTAL, CONFIG_NETLINK, CONFIG_RTNETLINK,
# CONFIG_IP_ADVANCED_ROUTER, CONFIG_IP_MULTIPLE_TABLES,
CONFIG_NET_SCHED,
# CONFIG_NET_SCH_CBQ, CONFIG_NET_SCH_TBF, CONFIG_NET_QOS,
# CONFIG_NET_ESTIMATOR, CONFIG_NET_CLS, CONFIG_NET_CLS_U32, and
# CONFIG_NET_POLICE.

# See --help for usage information. When run, any class-based
queueing
# parameters for the specified interface in your running kernel will
be
# erased, and a fresh rate-limiting setup will be created.

require 5.004; # my() in control structures
use Getopt::Long;
use POSIX qw(floor);
use Socket;

# If your iproute tools (ip and tc) aren't in any of these
directories, then
# add the appropriate directory.
$ENV{PATH} = '/sbin:/usr/sbin:/bin:/usr/bin';

sub usage ()
{
    print <<EOF;
    ratelimit v0.3, copyright (c) Zeus Technology Limited.

Usage: $0 [options]

Options are as follows (with defaults in brackets):

    --help                This help screen.
    --clean                Delete all queueing structures in the
kernel.
    --list                 List the current queueing structures.
    --stat[istics]        Print some queueing statistics.

    Commonly used options:
    --verbose              Print traffic control commands as
they are                  executed.
    --interface [eth0]    Interface to rate-limit.
    --bandwidth [100Mbit] Bandwidth of the physical device.
    --hosts []             Host(s) to rate-limit, separated by
commas.
    --ports []             Port range(s) to rate-limit. For
example:
    --rate [28800]         1024-2048,3000,5000-6000
    --burst [5]            Desired speed in bits per second.
    --direction []         Maximum burst size in packets.
destination ports        Whether to limit source or
"both" or                 (use "source" or "dest[ination]"; if
                           omitted, limits both).

    Rarely used options:
    --mtu [1514]           Maximum transfer unit (default is
Ethernet).
    --avpkt [1200]        Average packet size (default should
be OK for                 be OK for
    --mpu [64]             Ethernet).
smallest packet           Minimum policing unit (set to
                           transferred).

EOF
    exit 0;
}

my %options = (
    help          => 0,
    clean         => 0,
    list          => 0,
    statistics    => 0,

```

```

        verbose          => 0,
        interface        => 'eth0',
        bandwidth        => '100Mbit',
        hosts            => '',
        ports            => '',
        rate             => 28800,
        burst            => 5,
        direction        => '',

        mtu              => 1514,
        avpkt            => 1200,
        mpu              => 64,
    );

    GetOptions( \%options,
        'help',
        'clean',
        'list',
        'statistics|stats',
        'verbose|v!',
        'interface|if=s',
        'bandwidth|bw=s',
        'hosts|h=s',
        'ports|p=s',
        'rate|r=i',
        'burst|b=i',
        'direction|d=s',
        'mtu=i',
        'avpkt=i',
        'mpu=i',
    );

    if( $options{help} ) {
        usage();
    }

    my( $source, $dest );
    if( $options{direction} =~ /^(?:source|both)?$/ ) {
        $source = 1;
    }
    if( $options{direction} =~ /^(?:dest.*|both)?$/ ) {
        $dest = 1;
    }
    unless( $source or $dest ) {
        usage();
    }

    if( not $options{clean} and not $options{list} and not
        $options{statistics}
        and defined $ARGV[0] ) {
        for( $ARGV[0] ) {
            if ( /^clean$/ )           { $options{clean} = 1; }
            elsif( /^list$/ )         { $options{list} = 1; }
            elsif( /^stat(?:istic)?s$/ ) { $options{statistics} = 1; }
        }
    }

    my @hosts = map {
        my $addr = inet_aton($_); defined($addr) ? inet_ntoa($addr) : ()
    } split /,/, $options{hosts};
    my @ports = map { /(.*)-(.*)/ ? ($1..$2) : $_ } split /,/,
        $options{ports};

    my $ratebyburst = $options{rate} * $options{burst} / 8;

    sub runtc ($)
    {
        print STDERR "$_[0]\n" if $options{verbose};
        # fork() and exec() explicitly, since system() traps SIGINT.
        my $pid = fork;
        if( $pid ) {
            waitpid( $pid, 0 );
        } else {
            die "couldn't fork: $!" unless defined $pid;
            exec 'tc', split ' ', $_[0];
            die "couldn't exec tc: $!";
        }
    }

```

```

    }
}

if( $options{list} ) {
    print "Queueing disciplines:\n";
    runc "qdisc list dev $options{interface}";
    print "\nClasses:\n";
    runc "class list dev $options{interface}";
    print "\nFilters:\n";
    runc "filter list dev $options{interface}";
    print "\n";
    exit;
} elseif( $options{statistics} ) {
    print "Queueing disciplines:\n";
    runc "-s qdisc list dev $options{interface}";
    print "\nClasses:\n";
    runc "-s class list dev $options{interface}";
    print "\nFilters:\n";
    runc "-s filter list dev $options{interface}";
    print "\n";
    exit;
}

if( $> != 0 ) {
    print STDERR "Traffic control may only be manipulated by
root.\n";
    exit 1;
}

runc "qdisc del dev $options{interface} root";

exit if $options{clean};    # That's all we want to do.

runc "qdisc add dev $options{interface} root handle 1: " .
    "cbq bandwidth $options{bandwidth} allot $options{mtu} cell 8 "
    .
    "avpkt $options{avpkt} mpu $options{mpu}";

# Hashtable for source port filtering.
runc "filter add dev $options{interface} parent 1:0 protocol ip prio
5 " .
    "handle 1: u32 divisor 256";
# Hashtable for destination port filtering.
runc "filter add dev $options{interface} parent 1:0 protocol ip prio
5 " .
    "handle 2: u32 divisor 256";
# Hashtable for source and destination host filtering.
runc "filter add dev $options{interface} parent 1:0 protocol ip prio
5 " .
    "handle 801: u32 divisor 256";

# Match the protocol (here, TCP).
runc "filter add dev $options{interface} parent 1:0 prio 5 handle
::1 u32 " .
    "ht 800:: match ip nofrag offset mask 0x0f00 shift 6 " .
    "hashkey mask 0x00ff0000 at 8 link 801:";
# Key the hash table on (a) the source host and (b) the destination
host.
runc "filter add dev $options{interface} parent 1:0 prio 5 handle
::1 u32 " .
    "ht 801:6: match ip nofrag offset mask 0x0f00 shift 6 " .
    "hashkey mask 0x000000ff at 12 link 1:"
    if $source;
runc "filter add dev $options{interface} parent 1:0 prio 5 handle
::1 u32 " .
    "ht 801:6: match ip nofrag offset mask 0x0f00 shift 6 " .
    "hashkey mask 0x000000ff at 16 link 2:"
    if $dest;

runc "class add dev $options{interface} parent 1:0 classid 1:1 " .
    "cbq bandwidth $options{bandwidth} rate $options{bandwidth} " .
    "weight 1 avpkt $options{avpkt} prio 0 allot $options{mtu}";

for my $host ( @hosts ) {
    my $hashkey = sprintf '%x', ( ( unpack 'N', inet_aton($host) ) &
255 );
    for my $port ( @ports ) {

```


Zeus Web Server Configuration

The Zeus web server's global.cfg file was as follows:

```
errlog %zeushome%/web/log/errors
gid daemon
uid daemon
controlport 9080
controlallow 127.0.0.1
tuning!modules!stats!enabled no
tuning!sendfile yes
tuning!sendfile_reservefd 100
tuning!sendfile_minsize 1
tuning!listen_queue_size 32768
tuning!maxfds 65536
tuning!so_wbuff_size 65536
tuning!softservers no
tuning!keepalive_max 32768
tuning!keepalive_timeout 28800
tuning!num_children 4
```

The ZWS virtual server under test was configured as follows:

```
dnslookup no
env!inherited!PATH
env!inherited!TZ
modules!access!enabled no
modules!access!ldap!enabled no
modules!cgi!allowanywhere yes
modules!cgi!allowcmd yes
modules!cgi!autoid no
modules!cgi!chroot no
modules!cgi!enabled no
modules!cgi!ulimitas 0
modules!cgi!ulimitcpu 0
modules!cgi!ulimitdata 0
modules!cgi!ulimitnproc 0
modules!dirlist!enabled no
modules!distributed!enabled no
modules!distributed!servlet!prefix /servlet
modules!distributed!sockd!prefix /distributed
modules!errors!directory %zeushome%/web/etc/errpages
modules!errors!enabled yes
modules!fastcgi!enabled no
modules!frontpage!enabled no
modules!frontpage!fphome /usr/local/frontpage
modules!get!enabled yes
modules!htaccess!accessfilename .htaccess
modules!htaccess!enabled no
modules!imagemap!enabled yes
modules!index!enabled yes
modules!index!files index.html, index.htm, index.shtml,
index.cgi
modules!isapi!enabled no
modules!jserv!enabled no
modules!log!enabled no
modules!log!format %h %l %u %t "%r" %s %b
modules!map!alias_types!cgi application/x-httpd-cgi
modules!map!alias_types!fastcgi application/x-httpd-fcgi
modules!map!alias_types!isapi application/x-httpd-isapi
modules!map!alias_types!simple application/x-httpd-alias
modules!map!enabled yes
modules!map!homedir!dir public_html
modules!map!homedir!enabled yes
modules!map!homedir!file /etc/passwd
modules!map!homedir!ldap_filter cn=%u
modules!mime!default text/plain
modules!mime!enabled yes
modules!mime!types!Z application/octet-stream
```

```
modules!mime!types!api application/x-httpd-isapi
modules!mime!types!asis httpd/send-as-is
modules!mime!types!cgi application/x-httpd-cgi
modules!mime!types!exe application/octet-stream
modules!mime!types!gif image/gif
modules!mime!types!gz application/octet-stream
modules!mime!types!htm text/html
modules!mime!types!html text/html
modules!mime!types!jpeg image/jpeg
modules!mime!types!jpg image/jpeg
modules!mime!types!map application/x-httpd-imap
modules!mime!types!mov video/quicktime
modules!mime!types!mpeg video/mpeg
modules!mime!types!pdf application/pdf
modules!mime!types!shtml text/x-server-parsed-html
modules!mime!types!tar application/octet-stream
modules!mime!types!tgz application/octet-stream
modules!mime!types!txt text/plain
modules!mime!types!vrml x-world/x-vrml
modules!mime!types!wbmp image/vnd.wap.wbmp
modules!mime!types!wml text/vnd.wap.wml
modules!mime!types!wmlc application/vnd.wap.wmlc
modules!mime!types!wmls text/vnd.wap.wmlscript
modules!mime!types!wmlsc application/vnd.wap.wmlscriptc
modules!nsapi!config_root %zeushome%/ns-config
modules!nsapi!enabled no
modules!put!enabled no
modules!put!save_as_user no
modules!search!enabled no
modules!search!input_template
%zeushome%/web/etc/search_form.html
modules!search!output_template
%zeushome%/web/etc/search_output.html
modules!search!url /search
modules!spelling!enabled no
modules!stats!enabled no
modules!subserver!enabled no
modules!throttle!enabled no
modules!throttle!limit 128000
modules!throttle!maxconns 50
modules!throttle!subserver no
port 8042
security!client_cert 0
security!enabled no
ip_name goliath-giga
docroot /space/benchmarking/pages
aliases
bindaddr
bindport
errlog
webmaster
```

Apache Web Server Configuration

The Apache web server's httpd.conf file was as follows. Much of this configuration is the default as shipped with Apache 1.3.12.

```
ServerType standalone
ServerRoot "/usr/local/apache"
PidFile /usr/local/apache/logs/httpd.pid
ScoreBoardFile /usr/local/apache/logs/httpd.scoreboard
Timeout 300

KeepAlive On
MaxKeepAliveRequests 0
KeepAliveTimeout 28800

MinSpareServers 5
MaxSpareServers 50
StartServers 20
MaxClients 256
MaxRequestsPerChild 0

Port 8043
User www
ServerAdmin colinw@zeus.com
ServerName goliath-giga.cam.zeus.com
DocumentRoot "/space/benchmarking/pages"

<Directory />
    Options FollowSymLinks
    AllowOverride None
</Directory>

<Directory "/space/benchmarking/pages">
    Options Indexes FollowSymLinks MultiViews
    AllowOverride None
    Order allow,deny
    Allow from all
</Directory>

<IfModule mod_userdir.c>
    UserDir public_html
</IfModule>

<IfModule mod_dir.c>
    DirectoryIndex index.html
</IfModule>

AccessFileName .htaccess
<Files ~ "^\.ht">
    Order allow,deny
    Deny from all
</Files>

UseCanonicalName On

<IfModule mod_mime.c>
    TypesConfig /usr/local/apache/conf/mime.types
</IfModule>
DefaultType text/plain
<IfModule mod_mime_magic.c>
    MIMEMagicFile /usr/local/apache/conf/magic
</IfModule>

HostnameLookups Off

ErrorLog /usr/local/apache/logs/error_log
LogLevel warn
```

ServerSignature On

```

<IfModule mod_alias.c>
  Alias /icons/ "/usr/local/apache/icons/"
  <Directory "/usr/local/apache/icons">
    Options Indexes MultiViews
    AllowOverride None
    Order allow,deny
    Allow from all
  </Directory>
  ScriptAlias /cgi-bin/ "/usr/local/apache/cgi-bin/"
  <Directory "/usr/local/apache/cgi-bin">
    AllowOverride None
    Options None
    Order allow,deny
    Allow from all
  </Directory>
</IfModule>

<IfModule mod_autoindex.c>
  IndexOptions FancyIndexing
  AddIconByEncoding (CMP,/icons/compressed.gif) x-compress
  x-gzip
  AddIconByType (TXT,/icons/text.gif) text/*
  AddIconByType (IMG,/icons/image2.gif) image/*
  AddIconByType (SND,/icons/sound2.gif) audio/*
  AddIconByType (VID,/icons/movie.gif) video/*
  AddIcon /icons/binary.gif .bin .exe
  AddIcon /icons/binhex.gif .hqx
  AddIcon /icons/tar.gif .tar
  AddIcon /icons/world2.gif .wrl .wrl.gz .vrm .iv
  AddIcon /icons/compressed.gif .Z .z .tgz .gz .zip
  AddIcon /icons/a.gif .ps .ai .eps
  AddIcon /icons/layout.gif .html .shtml .htm .pdf
  AddIcon /icons/text.gif .txt
  AddIcon /icons/c.gif .c
  AddIcon /icons/p.gif .pl .py
  AddIcon /icons/f.gif .for
  AddIcon /icons/dvi.gif .dvi
  AddIcon /icons/uuencoded.gif .uu
  AddIcon /icons/script.gif .conf .sh .shar .csh .ksh .tcl
  AddIcon /icons/tex.gif .tex
  AddIcon /icons/bomb.gif core
  AddIcon /icons/back.gif ..
  AddIcon /icons/hand.right.gif README
  AddIcon /icons/folder.gif ^^DIRECTORY^^
  AddIcon /icons/blank.gif ^^BLANKICON^^

  DefaultIcon /icons/unknown.gif
  ReadmeName README
  HeaderName HEADER
  IndexIgnore .??* *~ *# HEADER* README* RCS CVS *,v *,t
</IfModule>

<IfModule mod_mime.c>
  AddEncoding x-compress Z
  AddEncoding x-gzip gz tgz
  AddLanguage da .dk
  AddLanguage nl .nl
  AddLanguage en .en
  AddLanguage et .ee
  AddLanguage fr .fr
  AddLanguage de .de
  AddLanguage el .el
  AddLanguage it .it
  AddLanguage ja .ja
  AddCharset ISO-2022-JP .jis
  AddLanguage pl .po

```

```
AddCharset ISO-8859-2 .iso-pl
AddLanguage pt .pt
AddLanguage pt-br .pt-br
AddLanguage ltz .lu
AddLanguage ca .ca
AddLanguage es .es
AddLanguage sv .se
AddLanguage cz .cz
<IfModule mod_negotiation.c>
    LanguagePriority en da nl et fr de el it ja pl pt pt-
br ltz ca es sv
</IfModule>
    AddType application/x-tar .tgz
</IfModule>

<IfModule mod_setenvif.c>
    BrowserMatch "Mozilla/2" nokeepalive
    BrowserMatch "MSIE 4\.0b2;" nokeepalive downgrade-1.0
force-response-1.0
    BrowserMatch "RealPlayer 4\.0" force-response-1.0
    BrowserMatch "Java/1\.0" force-response-1.0
    BrowserMatch "JDK/1\.0" force-response-1.0
</IfModule>
```

Router Kernel Configuration

Each router was running a Linux 2.2.15 kernel. The kernel was compiled with those options necessary to support the hardware and provide basic networking. In addition, the following options related specifically to the techniques discussed in this paper were set:

```

CONFIG_EXPERIMENTAL=y (Prompt for development and/or incomplete
code/drivers)

CONFIG_NETLINK=y (Kernel/User netlink socket)
CONFIG_RTNETLINK=y (Routing messages)
CONFIG_NETLINK_DEV=y (Netlink device emulation)
CONFIG_FIREWALL=y (Network firewalls)
CONFIG_IP_ADVANCED_ROUTER=y (IP: advanced router)
CONFIG_IP_MULTIPLE_TABLES=y (IP: policy routing)
CONFIG_IP_ROUTE_MULTIPATH=y (IP: equal cost multipath)
CONFIG_IP_ROUTE_TOS=y (IP: use TOS value as routing key)
CONFIG_IP_ROUTE_VERBOSE=y (IP: verbose route monitoring)
CONFIG_IP_ROUTE_LARGE_TABLES=y (IP: large routing tables)
CONFIG_IP_ROUTE_NAT=y (IP: fast network address translation)
CONFIG_IP_FIREWALL=y (IP: firewalling)
CONFIG_IP_FIREWALL_NETLINK=y (IP: firewall packet netlink device)
CONFIG_IP_ROUTE_FWMARK=y (IP: use FWMARK value as routing key)
CONFIG_IP_TRANSPARENT_PROXY=y (IP: transparent proxy support)
CONFIG_IP_ALIAS=y (IP: aliasing support)
CONFIG_SKB_LARGE=y (IP: Allow large windows)

CONFIG_NET_SCHED=y (QoS and/or fair queueing)
CONFIG_NET_SCH_CBQ=m (CBQ packet scheduler)
CONFIG_NET_SCH_CSZ=m (CSZ packet scheduler)
CONFIG_NET_SCH_PRIO=m (The simplest PRIO pseudoscheduler)
CONFIG_NET_SCH_RED=m (RED queue)
CONFIG_NET_SCH_SFQ=m (SFQ queue)
CONFIG_NET_SCH_TEQL=m (TEQL queue)
CONFIG_NET_SCH_TBF=m (TBF queue)
CONFIG_NET_QOS=y (QoS support)
CONFIG_NET_ESTIMATOR=y (Rate estimator)
CONFIG_NET_CLS=y (Packet classifier API)
CONFIG_NET_CLS_ROUTE4=m (Routing table based classifier)
CONFIG_NET_CLS_FW=m (Firewall based classifier)
CONFIG_NET_CLS_U32=m (U32 classifier)
CONFIG_NET_CLS_RSVP=m (Special RSVP classifier)
CONFIG_NET_CLS_POLICE=y (Ingress traffic policing)

```