# The Effects of Wide-Area Conditions on WWW Server Performance

Erich M. Nahum[1] Marcel-Cătălin Roşu[1] Srinivasan Seshan[2] Jussara Almeida[3]

Networked Systems Department[1]
IBM T.J. Watson Research Ctr.
Yorktown Heights NY
{nahum,rosu}@watson.ibm.com

Dept. of Computer Science[2]
Carnegie-Mellon Univ.
Pittsburgh PA
srini@cmu.edu

Dept. of Computer Science[3]
University of Wisconsin
Madison WI
jussara@cs.wisc.edu

**Abstract**

WWW workload generators are used to evaluate web server performance, and thus have a large impact on what performance optimizations are applied to servers. However, current benchmarks ignore a crucial component: how these servers perform in the environment in which they are intended to be used, namely the wide-area Internet.

This paper shows how WAN conditions can affect WWW server performance. We examine these effects using an experimental testbed which emulates WAN characteristics in a live setting, by introducing factors such as delay and packet loss in a controlled and reproducible fashion. We study how these factors interact with the host TCP implementation and what influence they have on web server performance. We demonstrate that when more realistic wide-area conditions are introduced, servers exhibit very different performance properties and scaling behaviors, which are not exposed by existing benchmarks running on LANs. We show that observed throughputs can give misleading information about server performance, and thus find that maximum throughput, or capacity, is a more useful metric. We find that packet losses can reduce server capacity by as much as 50 percent and increase response time as seen by the client. We show that using TCP SACK can reduce client response time, without reducing server capacity.

## 1 Introduction

The phenomenal growth of the World-Wide Web is dramatically increasing the performance requirements for large-scale information servers. WWW server performance is thus a central issue in providing ubiquitous, reliable, and efficient information access. Web server performance is frequently evaluated using WWW workload generators, which consequently have a large impact on what performance optimizations are applied to servers. While much research has been done in producing more realistic workload generators and optimizing WWW server performance, a crucial component has usually been ignored: how these servers perform in the environment in which they are intended to be used, namely the wide-area Internet.

Many web servers are used for wide-area information dissemination on the global Internet, which has greatly varying bandwidths, round-trip times, and packet loss characteristics. To date, virtually all experimental WWW server performance evaluation has been done on high-speed LANs, which have very different performance characteristics from WANs. These LANs, typically switched 100 Mbps Ethernet, provide excellent network connectivity, with high bandwidth and almost no packet loss or re-ordering. Given that many web servers are deployed in the commercial Internet, these evaluations are not realistic in that they do not capture wide-area characteristics such as modem-connected clients, high packet loss, and large packet delay. Even corporate Intranets or VPNs, while administered by a single organization, are geographically distributed and thus have very different performance characteristics. Thus, when customers wish to compare different systems or perform WWW server capacity planning, they need to take WAN characteristics into account; otherwise they may draw misleading conclusions.

Evaluating how web server performance changes in the context of this more realistic WAN environment is the goal of our project, called WASP (Wide-Area Server Performance). We focus on three metrics of WWW server performance: observed throughput, response time as seen by a client, and maximum throughput or *capacity*. We are particularly interested in how WAN characteristics interact with the host TCP implementation, and their effects on the server. Issues we consider include:

- *How does server performance scale with load?*
- *How do packet delay and loss affect observed throughput?*
- *What is the impact of loss and delay on response time as seen by the client?*
- *How do TCP variants such as SACK and New Reno influence performance?*
- *How do delay and loss affect server capacity?*

We evaluate these effects using an experimental testbed, which emulates WAN characteristics in a live setting by introducing factors such as delay and packet loss in a controlled and reproducible fashion. The testbed consists of a cluster of PCs acting as clients, connected via a switched LAN to a high-performance web server.

We show that when more realistic wide-area conditions are introduced, servers exhibit very different performance properties and

scaling behaviors, which are not exposed by current benchmarks such as WebStone or SPECWeb. While it is easy to saturate a server when WAN conditions are ignored, driving a server to full capacity is much more difficult when delays and drops are introduced. Observed throughputs can give misleading information about server performance, and thus capacity is a more useful metric. We find that packet loss both reduces aggregate server capacity and increases response time as seen by the client. We demonstrate that while packet delay increases latency, it does not have a substantial effect on server capacity. We show that while traditional benchmarks expose little or no difference between different versions of TCP, such as SACK or New Reno, these newer versions of TCP can improve client response time in WANs, without any reduction in server capacity.

Current benchmarks do not expose *any* of these issues, and thus we claim WWW server benchmarks should incorporate WAN characteristics. This will not only make server benchmarks more realistic, but will provide greater incentive and guidance for developers to adopt optimizations that make more efficient use of the network.

The rest of this paper is organized as follows: Section 2 provides more background on WWW servers and reviews previous work. Section 3 presents the WASP architecture, and Section 4 presents our results in detail. Section 5 summarizes our conclusions and briefly discusses plans for future work.

## 2 Background and Related Work

The demand to improve Web server efficiency has existed for quite some time and, as a result, there have been many previous efforts to measure and characterize the performance of these servers. These efforts can be roughly divided into benchmarking studies and analyses of live servers.

### 2.1 Other Benchmarks

Most Web server benchmarks use a collection of virtual clients connected via a high speed LAN to the server being tested. These virtual clients are used to accurately recreate the workloads that typical busy Web servers experience. Previous efforts to accurately reproduce the typical workload on a server have concentrated on certain aspects of user request access patterns:

- *File size, type and popularity*. Authors of benchmarks such as SURGE [7] and SPECWeb96 [10] developed their file size and request distribution by analyzing the logs of many popular Web sites. As usage patterns have changed over time, some benchmarks such as SPECWeb99 [10] have been updated with more recent request distributions and the addition of dynamic content. Many benchmarks view request-related information as the key to understanding Web server performance and reproduce little else.

- *Request arrival and rate*. Most benchmarks attempt to stress the server by requesting objects as quickly as possible. However, some benchmarks attempt to recreate more realistic arrival distributions. The SURGE benchmark uses the concept of "user equivalents" to recreate the active and idle periods of typical users. Another benchmark, Scalable Client (s-client) [4], uses an "open-loop" architecture to produce a fixed request arrival rate regardless of server load. Both these approaches cause the server to deal with the bursty arrival of requests, which can significantly affect server performance.

- *Protocol version*. Some benchmarks, such as SPECWeb99, have included both HTTP 1.0 and 1.1 support. The differences between these protocols can result in very different server performance.

Some benchmarking efforts, as well as our own, have begun to incorporate network characteristics into server evaluation. For example, SPECWeb99 restricts the TCP maximum segment size (MSS) to be no larger than 1460 bytes. This reflects the fact that most clients on the Internet use a MSS of fewer than 1460 bytes and that larger MSS values in a testbed can inflate the benchmark results.

### 2.2 Analyzing Active Web Servers

Another approach to understanding Web server performance is to profile and analyze an actively used server. This approach was used to examine the 1994 California Election Web Site [19] and the 1996 Summer Olympics Web Site [30]. Each study analyzed one of the busiest web servers of its time to examine how the protocol stack dealt with typical Web traffic. The 1994 study identified performance problems with protocol control block (PCB) lookups and the handling of TCP connections in the TIME_WAIT state. The suggested fixes for these performance problems were quickly incorporated into operating systems. The 1996 study concentrated on TCP behavior and made suggestions to improve loss recovery. These analyses identified performance problems that are difficult to find with current benchmarks. Unfortunately, such studies are arduous to undertake since they require access to busy sites. The busiest servers are typically commercial, and thus have privacy and security concerns, making it difficult to gain access to them. Analyzing live servers is, unfortunately, rarely an available option for discovering problems in WWW server performance.

The Wide Area Web Measurement (WAWM) project [6, 8] takes a related approach in that it places virtual test clients in the wide area network. The clients send synthetic requests generated by SURGE to a remote server. This guarantees the incorporation of realistic network characteristics, but using an actual wide area network makes it difficult to control the experiment carefully.

More fundamentally, measuring live sites in the real Internet presents serious challenges to reproducibility, as one-time events are measured. Researchers cannot be certain what the traffic conditions are at the time of the event, since they do not have complete control over the server, the network, and the clients. Nor can they be certain that subsequent experiments have the same characteristics. In addition, "what if" studies may not be feasible since changes may affect the workload that was observed. The WASP project addresses this by recreating network characteristics in a controlled environment, thus allowing reproducibility and iterative analysis. Our approach enables experimentation with both existing and future network conditions, and evaluation of to-be-deployed server features in a controlled setting.

## 3 The WASP Environment

In this Section we present the WASP testbed. Before we describe our setup in Section 3.4, we provide the context of the hardware used, the operating systems, the WWW server software, and the web client workload generators. Section 3.5 presents our experimental methodology.

### 3.1 Hardware and Operating System Software

Our testbed consists of 8 client PCs, a gigabit switch, and a RISC-based server machine. Each client has a 500 MHz Pentium/III processor and 96 MB of RAM, and runs FreeBSD 3.3. Each client has a 100 Base-T Ethernet interface connected point-to-point full duplex with the switch. The server machine is an IBM RS/6000 43P model 260, which has a 200 MHz Power3 processor, 4 MB of L2 Cache RAM, and 256 MB of main memory. The server runs AIX 4.3.3, IBM's UNIX OS, and is connected to the switch via an Alteon gigabit Ethernet adaptor.

On the server OS, we extended the TCP/IP stack with our own SACK [18] and New Reno [14] implementations. The TCP/IP stack in AIX is derived from BSD 4.4. AIX has been previously refined to better handle large-scale workloads, for example to use hash tables for PCB lookup and to separately manage connections in the TIME_WAIT state [3, 31].

On the client OS, we incorporated Rizzo's SACK source code [27] from FreeBSD 2.6 into the 3.3 kernel, in order to interoperate with our server SACK implementation.

### 3.2 WWW Server Software

Many available web servers employ different architectures and optimizations and thus have different performance characteristics. In order to minimize any limitations of our study that might occur by being too tied to a specific implementation, we wished to examine WAN effects for both a general-purpose and a highly-optimized server. In addition, we wanted to have access to the source code for the servers. We decided to use two servers: Apache, as an example of a general-purpose server, and Flash, as an example of a highly-optimized one.

Apache is the well-known open-source general-purpose server found at `www.apache.org`. Netcraft [21] reports that Apache has the largest market share of all WWW servers, estimating that it is used by over 50 percent of web sites on the Internet. Apache is a process-based server, implemented in user space, forking several processes, which serially accept new connections. We use the latest stable code release at the time of this writing, version 1.3.17, which has several performance improvements over previous Apache releases, including use of `mmap()` and `writev()`.

Flash is a WWW server developed by Pai *et al.* [24] at Rice University. Flash is a single-threaded event-driven server that uses asynchronous I/O with the `select()` system call. Flash exploits most optimizations that are available to a user-space web server without modifying the operating system. It caches files in user space with `mmap()`, caches `stat()` information and URI lookups, and exploits `writev()`. Flash has been shown to be significantly faster than Apache. We extended Flash in several ways to obtain the maximum performance in our system [20]. First, we modified it to `poll()` rather than `select()`. Banga and Mogul [5] have shown that event-driven servers that use `select()`, such as Flash, can suffer performance problems when managing large numbers of active connections. To avoid these problems, we adapted Flash to use `poll()`, a similar function derived from SVR4, which is available on some forms of Unix, including AIX. We also modified Flash to exploit the zero-copy `send_file()` system call available in AIX. Combined with the checksum offload on the Alteon gigabit adaptor, the server performs no unnecessary memory-touching operations when servicing requests on the fast path. Finally, we take advantage of the `send_file()` option, which allows consistently piggybacking the FIN bit on the last TCP segment of each conversation, improving performance for small transfers.

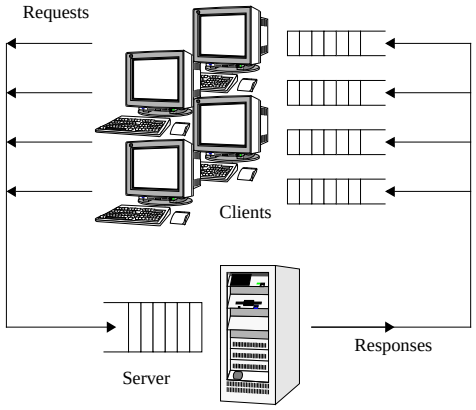


Figure 1: Workload Generation

### 3.3 WWW Client Workload Generator Software

We use two client HTTP workload generators to evaluate WWW server performance. We use s-client [4] as a micro-benchmark to generate large numbers of requests for the same single file on the server, which lets us see how performance changes as a function of the requested file size. We also use our own workload generator Waspclient as a macro-benchmark to measure more realistic aggregate behavior.

S-client is a lightweight, event-based request generator that uses a single process and the `select()` system call to manage a large number of concurrent active connections to the server. In our testbed, each client machine runs a single s-client process, which generates up to 2048 concurrent connections each. Figure 1 shows a representation of the workload architecture. For our experiments, we do not use the "open loop" facility of s-client. Each client makes a request to the server, waits for a response, and then immediately makes another request, throughout the duration of the test. The system is essentially a closed-form queuing network, where we can adjust the number of clients. By varying the number of concurrent connections, we can vary the load generated by the benchmark. This allows us to observe how the system scales as load is increased, and how its behavior changes as we vary parameters such as round-trip time and loss rate.

Waspclient is adapted from source code both from the s-client and the SURGE [7] benchmark. SURGE is a workload generator designed to capture various traffic characteristics of WWW sites, including heavy-tailed file and request size distributions, Zipf popularity of requested files, temporal locality of requested files, distributions of embedded references, and distribution of off times (or user think times). SURGE has the notion of a "user-equivalent", which represents the load produced by a single user. Larger numbers of users produce larger loads on the server. SURGE uses multiple processes and multiple threads, which do not scale as well as the single-process event-driven model in s-client. Hence, we found that it could not generate sufficient load to fully saturate our server. However, SURGE provides a more realistic workload and thus we wished to employ it. Since the traffic generation models are orthogonal to the implementation method, we imported the SURGE code (version 1.1, dated 4/98) onto the s-client event-driven architecture. The only feature we did not incorporate from SURGE was the user think time. Thus, each "user-equivalent" has no think time before requesting the next web page. While this may sacrifice some realism by reducing the burstiness of the request arrival process, we found that including user think times slowed the load generation

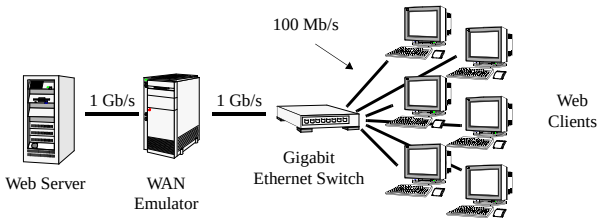


Figure 2: Centralized Approach

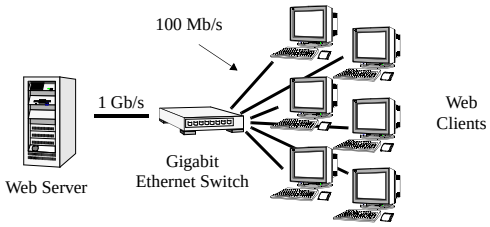


Figure 3: WASP Approach

too much for our purposes. We used the standard SURGE configuration parameters, e.g., using 2000 files for Zipf popularity and "optimal file/request matching" over the entire request distribution.

### 3.4 The WASP Environment

The WASP environment is designed to emulate WAN characteristics by dropping or delaying packets. The goal is not to reproduce a *specific* web site or network, but rather to generate *more realistic* traffic, in order to exercise a server in a way more useful for capacity planning. A central feature of our system is that it is isolated and reproducible, whereas the actual Internet is continually changing. An experiment run today on the Internet may not behave the same way the next week, month, or year in the future. Our system allows repeated experimentation, which lets us expose and diagnose performance issues.

Reproducing a specific web site perfectly would be effectively impossible. It would be necessary to replicate all the characteristics of the clients, including the browser behavior; the network, such as number of hops, bandwidths, queue sizes, and cross traffic; and the server itself, for example the content, the server software configuration, and the hardware platform. In addition, due to the size and heterogeneity of the Internet, any *one* instance of a site or network is probably not representative. For example, a large hosting center close to the MAE West NAP in California will most likely see different round-trip times to its clients than will a small company web site hosted over DSL in the Bronx.

Instead, we wish to examine a *range* of parameters that we believe are more realistic across most servers. We do a sensitivity analysis of servers to conditions rather than reproduce a specific point in the spectrum. Thus, we vary a parameter such as packet delay from 0 to 400 ms. rather than choosing a particular value, and then try to show what the impact of the delay is on a metric such as server throughput. In addition, our system has the advantage that it is *configurable*. Web hosting operators can parameterize the system based on measurements from their own sites and change them over time as necessary. The advantage of our system is that it allows iterative analysis using the *same* conditions. The disadvantage is that our emulated conditions may not be as realistic as desired. However, we believe the results are much *more* realistic than without any WAN conditions, and again, can be refined over time as better understanding of the real Internet is gained.

Previous approaches [4] to introducing packet drops and delays have relied on adding a single separate machine to the infrastructure to act as a WAN emulator, as shown in Figure 2. This machine is sometimes referred to as a *delay router*. Since all the WAN functionality is centralized in this single machine, we call this a *centralized approach*. This approach has the advantage that it requires no modifications to the client or server systems, which may

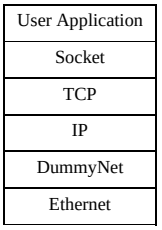


Figure 4: Dummynet Architecture

be necessary if source code is unavailable. While this approach is useful for protocol correctness testing, we have found that it not scalable enough for WWW server performance evaluation, given the high loads generated by the client machines. We describe this issue in more detail in Section 3.6, after we describe our experimental methodology in Section 3.5.

Instead, we use the configuration shown in Figure 3. Here, packet delay and loss is introduced on the clients directly, using the Dummynet [28] shim layer architecture in FreeBSD, shown in Figure 4. Dummynet resides below the protocol stack, and thus is transparent to the application. Dummynet applies *filter rules* to packets traversing the stack, and invokes dropping or delay functions if a packet *matches* a filter. Filters can match on a granularity as coarse-grained as a subnet or as fine-grained as a port number.

In order to delay packets, Dummynet uses the FreeBSD `timeout()` facility to schedule transmission of delayed packets. While the default period of this timer is 10 ms, we recompiled the kernel with `HZ = 1000` to get a finer-grained millisecond timer resolution. We were concerned that the 10 ms timer might affect our results, but found that this change made no difference. During our experiments, our client machines are never more that 50 percent utilized. However, we kept the clock at this resolution anyway, since our clients have many cycles to spare. While it is true that that packets may be delayed up to 1 ms longer than requested, we believe that this does not significantly disturb our experiments, since we examine delays from 50 to 400 ms. When Dummynet is not configured to add delay, the `timeout()` facility is bypassed; i.e., packets are sent directly.

To drop packets, Dummynet uses a uniform random loss model that can be configured to a specified probability. The loss model used in Dummynet is an *independent* one, i.e., the online decision to drop a packet is made regardless of past decisions. While this is useful for testing protocol correctness, it is most likely not a realistic portrayal of actual loss behavior in the Internet.

Packet loss is a complex phenomenon still being explored by

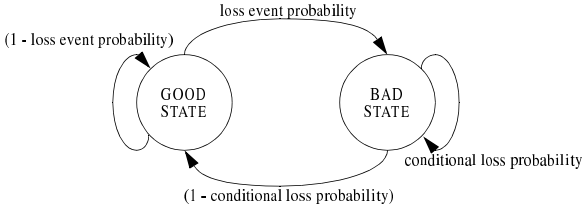


Figure 5: 2-State loss model

| Loss Prob. | Prob. leaving "good" state | Prob. leaving "bad" state |
|---|---|---|
| .00 | .00 | N/A |
| .03 | .02 | .65 |
| .06 | .04 | .65 |
| .09 | .07 | .65 |

Table 1: Transition probabilities

the research community. Several studies have shown that loss often occurs in *bursts* on the Internet [26, 29, 32]. The intuition is that routers drop packets when resources are scarce, and that resources tend to remain scarce for some finite period of time. Both Paxson [26] and Rubenstein *et al.* [29] observed that the conditional loss probability of the packet following a lost packet is much higher than the overall loss rate.

Motivated by these findings, we extended Dummynet to incorporate a two-state *dependent* loss model, shown pictorially in Figure 5. Each connection that passes through Dummynet is placed in either a "good" state or "bad" state. For connections in the "good" state, Dummynet forwards all packets, and for those in the "bad" state, all packets are dropped. The *transition rate* from the good state to the bad is set based on the rate that *loss events* occur, which is different from the overall packet loss rate. The likelihood of remaining in the "bad" state is the conditional loss probability, i.e., the probability that a subsequent packet is lost given that the previous packet is lost. The transition from the bad state to the good is thus the inverse of the conditional loss probability. This controls the duration of each loss event as well as the number of packets lost. Since the Dummynet extension maintains per-connection state, losses observed by a connection are correlated, but losses *across* connections are independent.

We chose the conditional loss probability in our tests to be 35 percent based on Paxson's observed measurements of the Internet. We calculated the other transition probabilities to produce specific overall loss rates. The values used are listed in Figure 1. For example, in the 6 percent loss configuration, a connection starts in the "good" state and has a 4 percent chance of having each packet dropped. If a packet is dropped, the connection transitions to the "bad" state, where its likelihood of having subsequent packets dropped is much higher, in this case 35 percent. When in the "bad" state and a packet is *not* dropped, the connection returns to the "good" state again.

NIST Net [9] is a tool similar to Dummynet, in that it is a transparent shim layer below the IP layer in the kernel, which can drop, delay, or bandwidth-limit packets. While Dummynet is distributed with FreeBSD, NIST Net is a separate package available for Linux. NIST Net has the disadvantage that it does not apply drops or delays to outgoing packets, but instead only to incoming packets. This means that in order to apply delays or losses symmetrically, a Linux box must be used as an intermediate router. Dummynet, on the other hand, applies rules to both incoming and outgoing packets, and thus can be used on the client. While we did experiment with NIST Net, we found Dummynet to be more appropriate for our experiments.

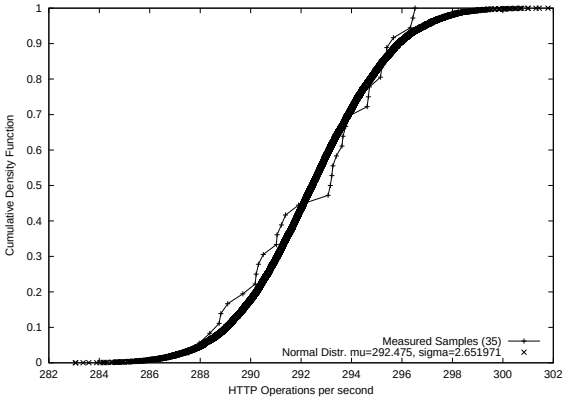


Figure 6: CDF of Waspclient real and synthetic samples

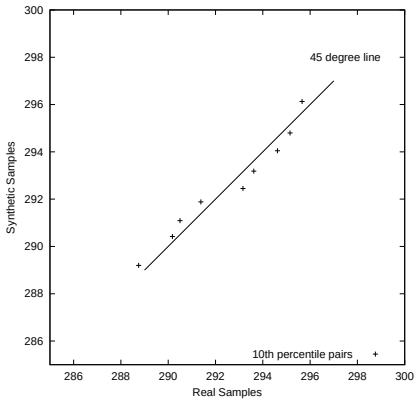


Figure 7: Waspclient probability plot using 10th percentiles

### 3.5 Experimental Methodology

An important consideration for our environment is attempting to be as scientifically reproducible as necessary, while balancing against the desire to examine as many configurations as possible. We thus took the following approach. In experiments using s-client, we measure requests for 3 representative file sizes: 1 KB to represent small files, 8 KB for an average file, and 64 KB for a large file. Each data point is the average of three runs, where each run is the average over a 60 second sampling interval after a 30 second warm-up. 64 KB experiments are sampled for 5 minutes, since 64 KB typically takes longer to transfer. In tests using Waspclient, we let each experiment run 10 minutes after a 30 second warm-up period, again taking the average of 3 experiments.

All graphs include 90 percent confidence intervals [16], calculated using the T distribution, which assumes that the underlying data distribution is normal. If the data is not normally distributed, then the variability reflected by the confidence intervals would not be appropriate. Given the breadth of the experiments that we do, we are not able to test this normality assumption for all data points. Instead, we check one "typical" data point by running one Wasp-

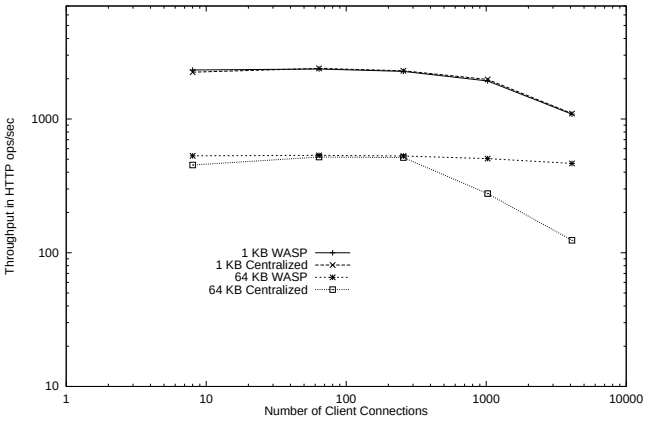


Figure 8: WASP Scalability

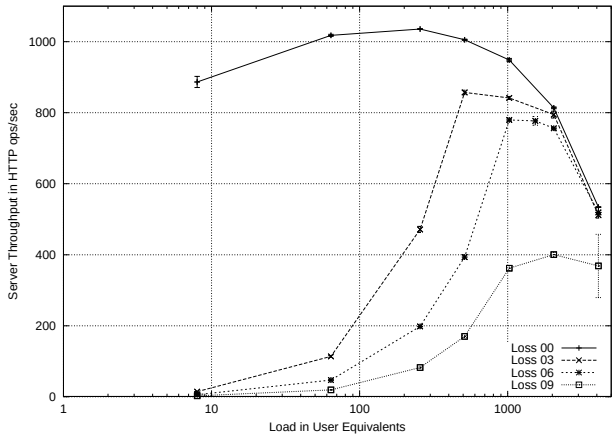


Figure 9: Flash Throughput vs. Dependent Loss Rate

client experiment and one s-client experiment 36 times each. In these experiments, the delay was set to 100 ms, the loss rate was 3 percent, and the server was 50 percent utilized. We then generated synthetic normal distributions using the same $\mu$ and $\sigma$ derived from the samples. Figure 6 shows the CDF from the real and synthetic samples from the Waspclient experiment. As can be seen, the normal distribution is a very tight fit. We also generated a 10th percentile probability plot [12], shown in Figure 7. If the two distributions match, the points will lie close to a 45-degree line, and we see that this is the case. Similar results were seen for the s-client samples, not shown due to space limitations. We thus assume that our other data points are also normally distributed. Since 3 samples are a valid parameter to the T distribution, we believe that the confidence intervals are appropriate.

### 3.6 WASP Scalability

The WASP environment scales with the number of load-generating machines, unlike the centralized approach, since the overhead of dropping or delaying packets is distributed across the clients. Figure 8 demonstrates the advantage of our approach. This graph shows server HTTP throughput as a function of the number of concurrent client requests generated using s-client. Four curves are shown: requests for 1 KB documents using the WASP approach, 1 KB documents using the centralized approach, 64 KB documents using the WASP approach, and 64 KB documents using the centralized approach. In this experiment, the delay router is a 450 MHz Pentium II running FreeBSD 3.3, with Dummynet enabled. However, Dummynet is configured to only match and forward packets, i.e., no delays or losses were introduced, and only a single rule for matching is used. While the centralized approach can handle the load for the 1 KB file requests, it cannot withstand the load for the 64 KB requests when large numbers of concurrent connections are used. The WASP environment, on the other hand, scales as long as the aggregate packet switching load is not greater than the gigabit switch's capacity. In the centralized approach above, the load cannot be greater than the router's software forwarding capacity, which is typically slower than switching. As will be seen in Section 4, very large numbers of concurrent requests are necessary to exercise the server when delays and losses are introduced. In addition, this test used only a single filter rule on the delay router. Large numbers of filters, such as might be needed to maintain per-connection state, will impose even more processing and memory requirements in the centralized case, and thus will inherently scale better when distributed across the client nodes.

At first glance this result could seem counter-intuitive. One might think that a router does less work than a WWW server, since all it does is forward packets, rather than handle HTTP requests, and thus would not saturate before the server does. However, since this is a software router on a PC with a PCI bus, it actually does more work on a per-packet basis. Each byte of a packet is copied across the memory bus twice: once from the inbound interface to the system memory, and again from system memory to the outbound interface. Thus the memory usage is typically twice that of the WWW server, which only copies a packet across the memory bus once, in the common case.

While a hardware-based router might be fast enough for our purposes, the advantage of our approach is that we use commodity hardware running public-domain software. The source code is freely available, can be customized and extended, and the system as a whole is generally less expensive. Similarly, another approach might be to add a dedicated software-based delay router in front of *each* client machine. This is part of the approach taken in Barford's thesis [6]. While this would also scale, it would require more hardware, thus being more expensive and more difficult to manage and administer. Our experience is that this is not necessary, since the client machines have sufficient idle cycles to handle the extra packet processing, as described in Section 3.4.

## 4 Results

In this section we present our results in detail. We show how packet loss and network delay affect server throughput, response time as seen by the client, and server capacity. In general, due to space limitations, we mostly present results using Waspclient to load the Flash server combined with the dependent loss model. However, we found similar trends and behavior using s-client, the independent loss model, and the Apache server.

### 4.1 Effects on Throughput

We begin by examining how increasing the loss rate affects the observed throughput of the web server. Figure 9 shows the measured throughput of the Flash server in HTTP operations/second as a function of load generated by the Waspclient workload generator

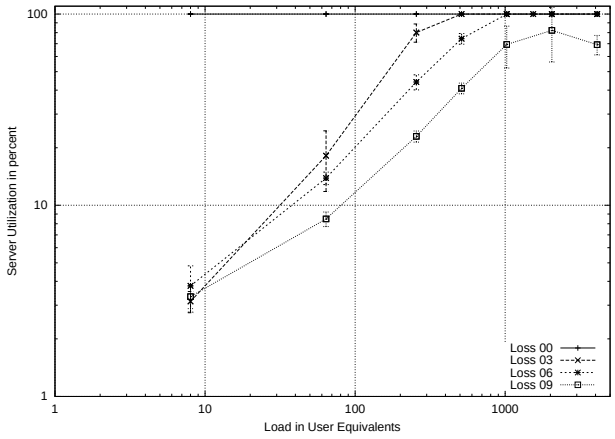


Figure 10: Flash Utilization vs. Dependent Loss Rate

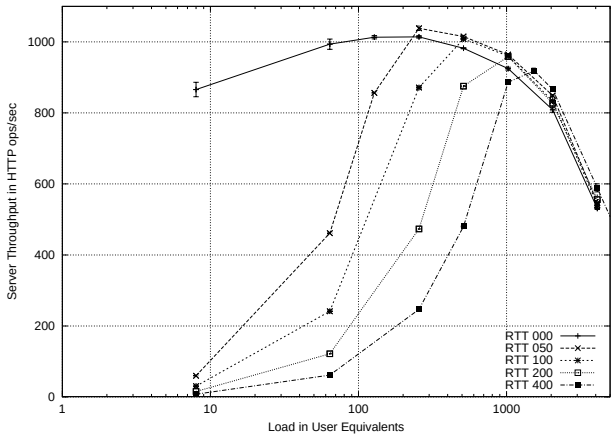


Figure 11: Flash Throughput vs. RTT

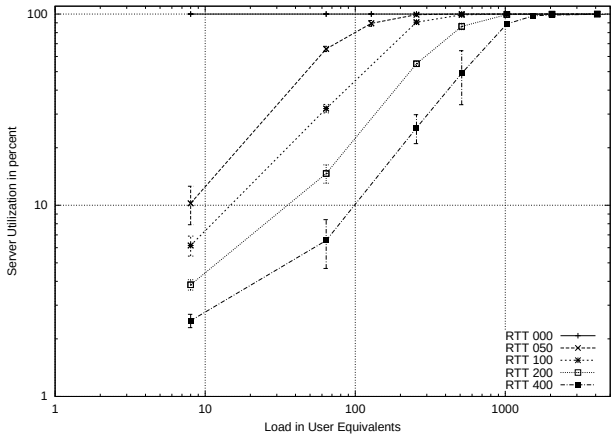


Figure 12: Flash Utilization vs. RTT

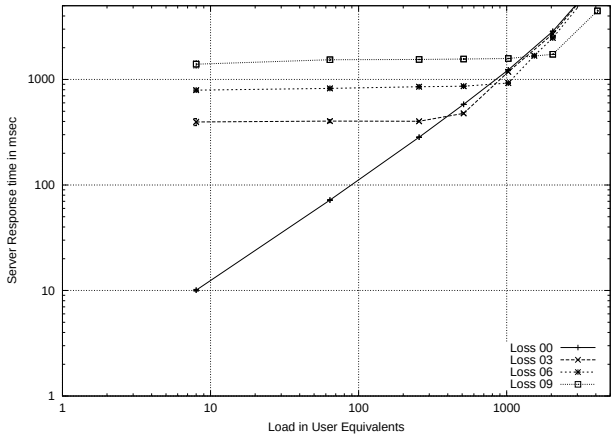


Figure 13: Flash Response Time vs. Dependent Loss Rate

in user equivalents (UE's). Four curves are plotted: with no loss, with 3 percent loss, with 6 percent, and with 9 percent. In this experiment, no delay is introduced; the point is to evaluate the impact of packet loss in isolation from other factors. Figure 10 shows the corresponding server machine CPU utilization during the test as measured by `iostat`. Note that on the throughput graph, the X axis is linear, whereas the Y axis is logarithmic. On the utilization graph, both axes are in log scale. As can be seen, the introduction of loss has a significant effect on the observed throughput in the system. Higher loss rates result in lower throughputs, regardless of load. In addition, we observe that increasing the load is necessary to bring the server to full saturation, as shown in Figure 10.

Next we examine how throughput is affected by the introduction of packet delay. Figure 11 shows the throughput of the Flash server, again as a function of load. Here five curves are plotted: with no delay, with 50 ms delay, with 100 ms, 200 ms, and 400 ms. In this test, no packet loss is introduced. Figure 12 shows the corresponding server CPU utilization measured during the test. Note again that on the X axis on the throughput graph is linear, while all other axes in the two Figures are logarithmic.

As with packet loss, introducing packet delay also has a significant effect on the observed throughput in the system, regardless of load. Higher round-trip times rates result in lower throughputs. Again we see that increasing the load is necessary to bring the server to full saturation, as depicted in Figure 12.

The cause of this behavior is the underlying TCP protocol's congestion avoidance mechanism. When loss or delay is introduced, it has the effect of 'slowing down' the request rate as observed by the server. Since this is a closed system, the arrival rate and the departure rate of jobs at the server are identical. Each "user equivalent" can only use a fixed number of connections, thus restricting the rate at which requests can be submitted to the server. In turn, each connection proceeds more slowly due to the loss or delay. This is because TCP's throughput is inversely proportional to the RTT and the square root of the loss. The simplified model for TCP throughput [13] states:

$$T \leq \frac{1.5\sqrt{2/3} * B}{R * \sqrt{p}}$$

where $T$ is throughput, $B$ is the maximum segment size (MSS), $R$ is the round-trip time, and $p$ is the loss rate. More elaborate models are available [22], but they all share this relationship between throughput, loss, and delay. Since latency is inversely proportional to throughput, for a fixed file size, the time to transfer a single file goes up. As the RTT or packet drop rate increases, a single connection thus puts *less load* on a server than a connection with a

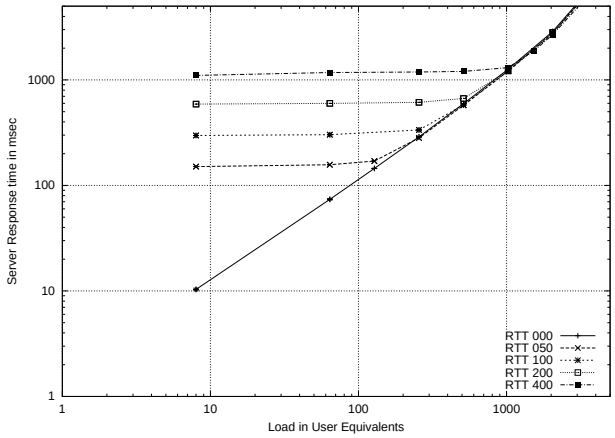


Figure 14: Flash Response Time vs. Delay

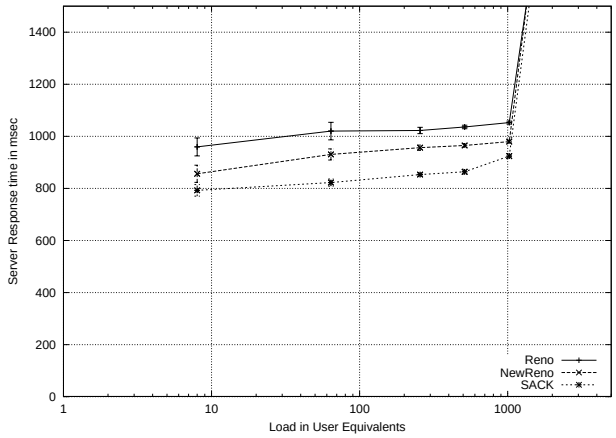


Figure 15: Response Times of TCP Variants (Flash, 6 % loss)

lower RTT. Similar trends were observed with other experiments, not shown due to space limitations, including tests using s-client, configurations with Apache, and experiments that used the independent loss model.

### 4.2 Effects on Response Time

Latency or response time, as seen by the client, is also an important metric in evaluating web server performance. Thus, we also wish to examine how response time is affected by both loss rate and round-trip times. Here the measured latency is the time between when the client application invokes the `connect()` call to the server and when the last byte of the response is delivered to the user application. It does not include DNS lookup times or page parsing delays. It does include, however, wire travel time and client network protocol processing time, in addition to the server processing time.

Figure 13 shows the Flash server's response time versus load when loss is introduced. Four curves are plotted, again one each for 0 percent, 3 percent, 6 percent, and 9 percent loss. Figure 14 shows Flash response time as packet delays are added. In both Figures, as the loss rate or packet delay increases, the average response time does as well. This is again due to TCP's throughput being inversely proportional to the square root of the loss rate [17, 22]. Since response time is typically inversely proportional to throughput, higher loss rates lead to higher response times.

Note that the no-delay and no-loss curves for each figure follow a diagonal line. This is because in these cases, the response time is limited by the server processing time. As load increases, jobs queue up and requests must wait longer for service. We see that in the curves when loss and delay are introduced, the load on the server is low and thus the client response times are network-limited. As the load in user-equivalents increases, each curve eventually intersects the diagonal line, where the server is fully saturated and thus queuing delay at the server starts contributing to response time. Again, we see the same trends, not shown due to space limitations, in similar experiments using s-client, for requests for different file sizes, with both Flash and Apache.

### 4.3 Effects of TCP Variants

A great deal of research has been accomplished studying different versions of TCP, such as Reno, SACK and New Reno. These TCP's have varying approaches to loss recovery and congestion management, and perform very differently when faced with losses, especially with respect to the round-trip time. Thus we thought it would be important to analyze these TCP variants using the WASP testbed.

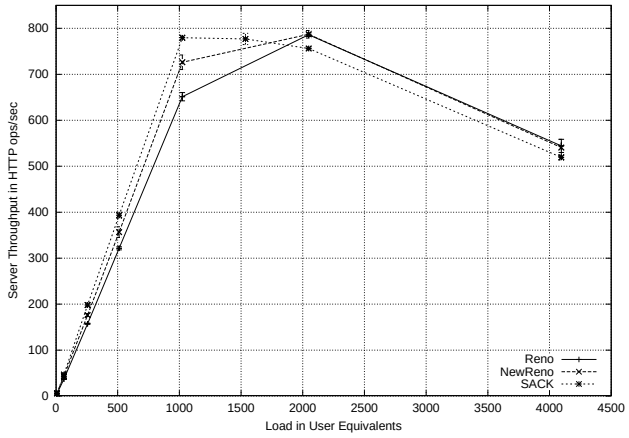


Figure 16: Throughputs of TCP Variants (Flash, 6 % loss)

Figure 15 shows the response time for the three TCP variants using an experiment with 6 percent loss and the Flash server. As can be seen, the latency measured by Waspclient using SACK is roughly 20 percent less than the time observed using Reno. New Reno is also better than Reno, in this case providing about a 10 percent improvement in response time.

This is because SACK, and to a lesser extent, New Reno, can recover lost packets more quickly than Reno. Reno can only retransmit a lost segment ('fill a hole') once every three round-trip times, whereas New Reno can fill one hole every RTT, and SACK can fill multiple holes per RTT. SACK is also more resilient to ACK loss on the return path. SACK can thus recover more quickly from loss, especially given large round-trip times. However, SACK can only provide an advantage over Reno when multiple losses happen within the current congestion window, which in turn only occurs when the window is large. Since TCP increases the congestion window via "slow-start", multiple rounds of packet exchanges must occur before the congestion window is large enough to allow SACK any loss recovery opportunities that are not available to Reno. This

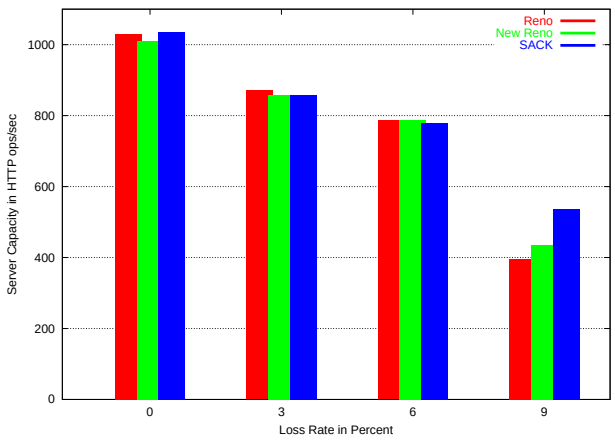


Figure 17: Flash Capacity vs. Dependent Loss Rate

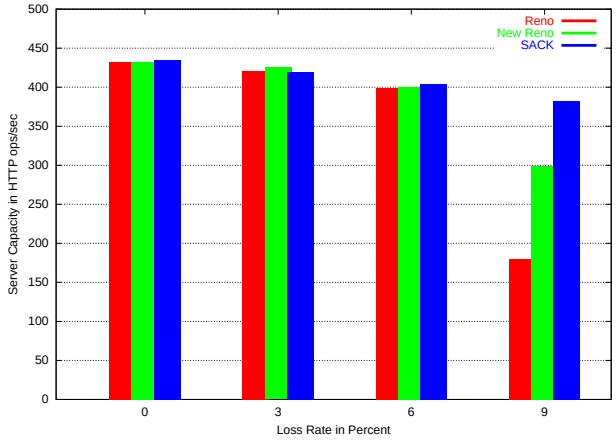


Figure 18: Apache Capacity vs. Dependent Loss Rate

means SACK makes no difference for short transfers. In experiments using s-client, not shown due to space limitations, SACK showed no improvements in tests requesting 1 KB files when loss was introduced. In experiments where requests were for 64 KB files, SACK's advantage over Reno was substantial. Since Wasp-client produces a set of requests for a wide range of file sizes, the workload generator exposes SACK's benefits.

Figure 16 shows the observed throughput for the three versions of TCP. Here we see, for smaller loads, SACK has the highest observed throughput, New Reno the next highest, and Reno the worst. As the loads are increased, each curve reaches a maximum throughput as the server is fully utilized, and then starts degrading with excess load. Note that these curves simply reflect the implications of the latencies from Figure 15. Since SACK has lower latency, it achieves greater throughput for a given load.

These results are important since these distinctions do not show up in conventional benchmarks where the WAN characteristics are ignored. Distinguishing these characteristics is necessary to provide the proper incentives for server manufacturers. For example, SpecWeb99 will not reward a vendor who supports SACK or New Reno in their TCP stack, yet in the real world their utility is clear, in terms of lower latencies seen by clients. New Reno is a sender-side only modification and thus can be put to use immediately. While SACK has the deployment difficulty in that it requires modification on both ends of the connection, this limitation is fading. Since Windows 98 supports SACK, a growing number of clients are able to negotiate SACK options, and in fact Allman [1] presents evidence that supports this. It is plainly beneficial to add SACK support to servers. Finally, while one might be concerned that SACK could incur extra processing overhead since it maintains additional per-connection state, we find that this is not an issue. We show in the next Section that adding SACK and New Reno support does not reduce a server's overall capacity.

### 4.4 Effects on Server Capacity

We saw in Section 4.1 that measuring throughput can be complex. Based on our experiences, we believe that simply measuring throughput without regard to server CPU utilization can cause misleading conclusions. For this reason, we examine our third metric, *capacity*.

Recall Figure 9 in Section 4.1, which shows how throughput varies with load for different loss rates. How then would capacity be defined? One answer might be to fix the load in number of users and report the throughput, but this method does not capture the whole picture. Consider the case where the system is under a load of 256 user-equivalents. Here the no-loss curve is maximized, with 1035 HTTP operations per second, whereas the 3 percent loss has 471 ops/sec and the 6 percent loss curve has about 198 HTTP ops/sec. As we saw from Figure 10, the server is not fully utilized in the 3 percent and 6 percent loss cases. Clearly, this is not a fair comparison, since the server has idle capacity in the experiments where loss is introduced. Now consider the case where the load is 1024 user-equivalents in Figure 9. Here, both the 6 percent and 3 percent loss curves are measuring fully utilized servers, whereas the 9 percent loss curve is not. But again, this comparison is not fair, since the no-delay experiment reflects the cost of jobs queued up at the server, which results in larger numbers of concurrent connections, whereas the 6 percent loss experiment does not. The proper comparison then is between the *maximums* of each curve, where each server is fully utilized, but is not penalized by the cost of queued jobs. We thus define capacity as the maximum over the curve, despite the fact that different numbers of concurrent users may be used to arrive at the maximum. In our experiments, the numbers of clients are varied from 8 to 4096 in powers of two, and then the maximum of all these points is chosen.

Figure 17 shows the server capacity versus the loss rate, using the three versions of TCP described earlier. As can be seen, the total capacity is reduced by up to 50 percent for high loss rates. Note also that the 3 TCP variants all have roughly the *same* capacity for a particular loss rate. The exception is in the 9 percent loss case, where SACK appears to outperform New Reno, which outperforms Reno. However, the picture here is more complicated because the server is not completely saturated. These high loss rates made it extremely difficult to fully utilize the server. In this case SACK's advantage is not in raw capacity but in its ability to let the server to process retransmissions more quickly and thus allow the server to effectively schedule tasks in a more work-conserving manner. Based on our other data, we believe that the actual capacity would be the same if the server could be driven to full utilization.

Figure 18 shows the capacity for the Apache server, again versus the loss rate. Note that the scale of the Y axis is not the same as in Figure 17, and that Apache is significantly slower than Flash, here by roughly a factor of 2. While the trend is not as severe as with Flash, there is still a noticeable reduction in capacity as the loss rate increases. Since Apache is less efficient than Flash, the

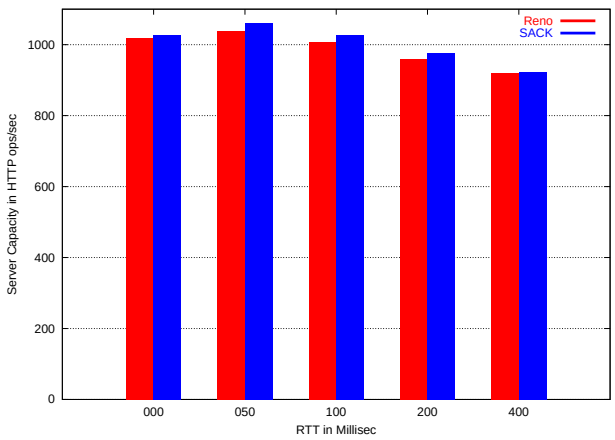


Figure 19: Flash Capacity vs. Delay

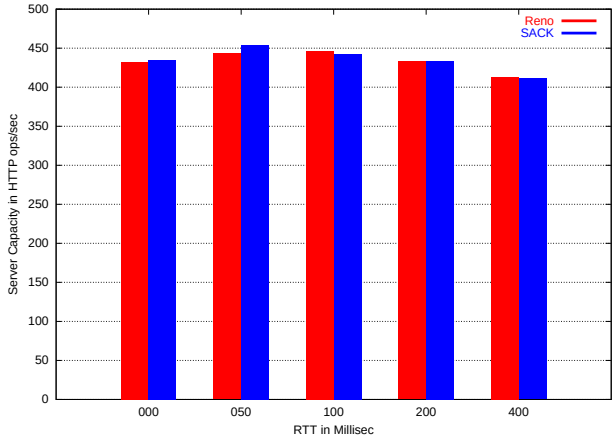


Figure 20: Apache Capacity vs. Delay

proportion of time spent in the network protocol stack is relatively smaller than is spent using Flash. Since loss increases the amount of work done by the protocol stack, the relative impact of loss on capacity is thus smaller for Apache. Note that we also see the same issue in the 9 percent loss case that we saw in Figure 17.

One might think that a server employing SACK would have greater capacity, since SACK is better at avoiding unneeded retransmissions, i.e., have better "goodput." In practice, we find that this is not a significant issue. Our intuition is that ultimately the server has some fixed capacity for sending packets, and that the choice of SACK or Reno simply influences how packets are distributed or scheduled across connections. In addition, given the conservative nature of TCP's loss recovery policies, unnecessary retransmissions are probably rare even in Reno. The advantage of SACK is thus to infer loss more quickly.

We now turn our attention to the effects of delay on server capacity. Figure 19 shows the Flash server capacity versus delay, where RTT is increased from zero up to 400 ms. Here we see a very slight falloff in capacity, and only with very large delays. Figure 20 shows the capacity versus delay using the Apache server, with the same trends. Again note the difference in the scale of the Y-axis between Figures 19 and 20. We see a very minor reduction in capacity with the Apache server as well.

Our RTT results are somewhat at odds with those from Banga and Druschel [4], who showed a more severe reduction in capacity with increasing round-trip times. Using delays up to 200 ms, they found that capacity shrunk up to 50 percent with Apache and up to 22 percent with Zeus. In our experiments, capacity was reduced only 5 percent, and only with 400 ms round-trip times. Without access to their exact setup, it is difficult to completely explain the difference, but we believe it is due to two factors. First, Banga and Druschel used an earlier version (1.2.4) of Apache, whereas here we use a more recent version of Apache (1.3.17), and use Flash, which is much more efficient with per-connection resources. Second, our AIX system employs the TIME_WAIT optimization described by Aron and Druschel [3], which was not present in the FreeBSD source used by Banga and Druschel at the time of their work. We believe consequently that state-of-the-art implementation techniques have greatly reduced, if not eliminated, this problem of larger RTT's reducing capacity.

## 5 Conclusions and Future Work

This paper has examined how WAN conditions can have a significant impact on WWW server performance, in ways that are not exposed by benchmarks run in a LAN setting. We summarize our conclusions as follows:

- *RTTs and losses matter.* We find that introducing packet delays and losses can have substantial effects on server throughput. Driving a system to saturation is more difficult when losses and delays are introduced, and the server exhibits different scaling behavior as load is increased. Different servers may react differently to these conditions, and thus it is important to introduce these characteristics into any test suite.

- *Packet losses reduce throughput and increase latency.* We show how increasing loss rates can lower server capacity by as much as 50 percent and increase the response times as seen by clients.

- *Packet delays increase latency but do not reduce throughput.* While large round-trip times slow responses as seen by clients, overall server capacity is unaffected.

- *Reno, SACK, and New Reno perform differently.* Different versions of TCP react very differently to packet losses. We show how using SACK or New Reno does not change server throughput, but can reduce client response time. We conclude that servers should deploy these TCP variants.

Again, the WASP environment allows researchers to study these effects and quantify their impact on performance. Based on these findings, our SACK and New Reno code has been incorporated into the AIX operating system. In addition, given that WWW proxies perform many of the same functions that web servers do, we anticipate that our results will apply to proxies as well.

This work has only begun to study the issue of WAN effects on servers. Many other aspects remain to be examined, e.g., dynamic content such as CGI or servlets, secure content over SSL/TLS, implementation issues involving code paths not exercised by current benchmarks, etc. In addition, the loss and delay models built into the system need to be refined based on a better understanding of how these phenomena occur in the wide-area Internet. The WASP

environment allows us to easily adapt our models and to study different networked server applications. We intend on pursuing many of these issues further.

## Acknowledgments

Many people improved this work through discussions or feedback on earlier drafts of this paper, including the anonymous referees, Paul Barford, Mark Crovella, Peter Druschel, Jennifer Rexford, and Anees Shaikh. Thanks to Venkat Venkatsubra for his assistance with AIX. Special thanks to those who gave us their source code: Mohit Aron for s-client, Paul Barford for SURGE, and Vivek Pai for Flash.

## References

[1] Mark Allman. A Web server's view of the transport layer. *Computer Communication Review*, 30(5), Oct 2000.

[2] Martin F. Arlitt and Carey L. Williamson. Internet Web servers: Workload characterization and performance implications. *IEEE/ACM Transactions on Networking*, 5(5):631–646, Oct 1997.

[3] Mohit Aron and Peter Druschel. TCP implementation enhancements for improving Webserver performance. Technical Report TR99-335, Rice University Computer Science Dept., July 1999.

[4] Gaurav Banga and Peter Druschel. Measuring the capacity of a Web server under realistic loads. *World Wide Web Journal (Special Issue on World Wide Web Characterization and Performance Evaluation)*, 2(1):69–83, May 1999.

[5] Gaurav Banga and Jeffrey C. Mogul. Scalable kernel performance for Internet servers under realistic loads. In *USENIX Annual Technical Conference*, New Orleans, Louisiana, June 1998.

[6] Paul Barford. *Modeling, Measurement and Performance of World Wide Web Transactions*. PhD thesis, Boston University, Boston, MA 02215, January 2001.

[7] Paul Barford and Mark Crovella. Generating representative Web workloads for network and server performance evaluation. In *Proceedings of the ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, Madison, WI, June 1998.

[8] Paul Barford and Mark Crovella. Critical path analysis of TCP transactions. In *ACM SIGCOMM Symposium on Communications Architectures and Protocols*, Stockhold, Sweden, August 2000.

[9] Mark Carson. NIST Net. Available at http://www.antd.nist.gov/nistnet.

[10] The Standard Performance Evaluation Corporation. SpecWeb96/SpecWeb99. http://www.spec.org/osg.

[11] Mark Crovella and Azer Bestavros. Self-similarity in World Wide Web traffic: Evidence and possible causes. *IEEE/ACM Transactions on Networking*, 5(6):835–846, Nov 1997.

[12] Jay L. Devore. *Probability and Statistics for Engineering and the Sciences (2nd Ed.)*. Brooks/Cole Publishing Co., 1987.

[13] Sally Floyd and Kevin Fall. Promoting the use of end-to-end congestion control in the Internet. *IEEE/ACM Trans. on Networking*, 7(4), August 1999.

[14] Sally Floyd and Tom Henderson. The NewReno modification to TCP's fast recovery algorithm. In *Network Information Center RFC 2582 (Experimental)*, April 1999.

[15] James C. Hu, Irfan Pyarali, and Douglas C. Schmidt. Measuring the impact of event dispatching and concurrency models on Web server performance over high-speed networks. In *Proceedings of the 2nd Global Internet Conference (held as part of GLOBECOM '97)*, Phoenix, AZ, Nov 1997.

[16] Raj Jain. *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, 1991.

[17] T. V. Lakshman and U. Madhow. The performance of TCP/IP for networks with high bandwidth-delay products and random loss. *IEEE/ACM Trans. on Networking*, 5(3), 1997.

[18] Matthew Mathis, Jamshid Mahdavi, Sally Floyd, and Allyn Romanow. TCP selective acknowledgment options. In *Network Information Center RFC 2018*, October 1996.

[19] Jeffrey C. Mogul. Network behavior of a busy Web server and its clients. Technical Report 95/5, Digital Equipment Corporation Western Research Lab, Palo Alto, CA, October 1995.

[20] Erich M. Nahum, Tsipora Barzilai, and Dilip Kandlur. Performance issues in WWW servers (extended abstract). In *Proceedings of the ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, Atlanta, GA, May 1999. An extended version has been accepted to appear in *IEEE/ACM Transactions on Networking.*

[21] Netcraft. The Netcraft WWW server survey. Available at http://www.netcraft.co.uk/Survey.

[22] Jitendra Padhye, Victor Firoiu, Don Towsley, and Jim Kurose. Modeling TCP throughput: A simple model and its empirical validation. In *ACM SIGCOMM Symposium on Communications Architectures and Protocols*, September 1998.

[23] Jitendra Padhye and Sally Floyd. Identifying the TCP behavior of Web servers. ICSI Tech Report 01-002, February 2001.

[24] Vivek Pai, Peter Druschel, and Willy Zwaenepoel. Flash: An efficient and portable Web server. In *USENIX Annual Technical Conference*, Monterey, CA, June 1999.

[25] Vivek S. Pai, Peter Druschel, and Willy Zwaenepoel. I/O Lite: A copy-free UNIX I/O system. In *3rd USENIX Symposium on Operating Systems Design and Implementation*, New Orleans, LA, February 1999.

[26] Vern Paxson. End-to-end Internet packet dynamics. *IEEE/ACM Transactions on Networking*, 7(3), June 1999.

[27] Luigi Rizzo. Issues in the implementation of selective acknowledgements for TCP. Draft report. Information available at http://info.iet.unipi.it/˜luigi/sack.html, Jan 1996.

[28] Luigi Rizzo. Dummynet: A simple approach to the evaluation of network protocols. *Computer Communication Review*, 27(2), Feb 1997.

[29] Dan Rubenstein, Jim Kurose, and Don Towsley. Detecting shared congestion of flows via end-to-end measurement. In *Proceedings of ACM SIGMETRICS 2000*, Santa Clara, CA, June 2000.

[30] Srinivasan Seshan, Hari Balakrishnan, Venkata N. Padmanabhan, Mark Stemm, and Randy Katz. TCP behavior of a busy Internet server: Analysis and improvements. In *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, San Francisco, CA, March 1998.

[31] W. Richard Stevens. Re: cleaning up TIME_WAIT states. In *End-to-end mailing list archives*, January 1997. Available at http://www.postel.org/mailman/listinfo/end2end-interest.

[32] Maya Yajnik, Sue B. Moon, Jim Kurose, and Don Towsley. Measurement and modelling of the temporal dependence in packet loss. In *Proceedings of 1999 IEEE INFOCOM*, New York, NY, March 1999.