A Performance Evaluation of Hyper Text Transfer Protocols

Paul Barford and Mark Crovella
Computer Science Department
Boston University
111 Cummington St, Boston, MA 02215

{barford,crovella,}@cs.bu.edu

BU-TR-98-016

October 27, 1998

Abstract

Version 1.1 of the Hyper Text Transfer Protocol (HTTP) was principally developed as a means for reducing both document transfer latency and network traffic. The rationale for the performance enhancements in HTTP/1.1 is based on the assumption that the network is the bottleneck in Web transactions. In practice, however, the Web server can be the primary source of document transfer latency. In this paper, we characterize and compare the performance of HTTP/1.0 and HTTP/1.1 in terms of throughput at the server and transfer latency at the client. Our approach is based on considering a broader set of bottlenecks in an HTTP transfer; we examine how bottlenecks in the network, CPU, and in the disk system affect the relative performance of HTTP/1.0 versus HTTP/1.1. We show that the network demands under HTTP/1.1 are somewhat lower than HTTP/1.0, and we quantify those differences in terms of packets transferred, server congestion window size and data bytes per packet. We show that when the CPU is the bottleneck, there is relatively little difference in performance between HTTP/1.0 and HTTP/1.1. Surprisingly, we show that when the disk system is the bottleneck, performance using HTTP/1.1 can be much worse than with HTTP/1.0. Based on these observations, we suggest a connection management policy for HTTP/1.1 that can improve throughput, decrease latency, and keep network traffic low when the disk system is the bottleneck.

1 Introduction

A source of ongoing frustration for users of the World Wide Web is the latency associated with data retrieval. Beyond a client's immediate connection to the Internet (which, in the case of slow modem users, can be the primary source of latency) there are essentially two possible sources of latency: the network and the server. Bottlenecks in the network exist due to congestion and/or limited bandwidth links along the path from client to server. Bottlenecks at the server occur when the server is heavily utilized either in the CPU or the disk system. When developing methods for improving performance in the Web, it is important to consider any potential solution from the perspectives of the client, network and server.

Initial work on improving Web performance focused on delays in the network as the principal source of latency for Web clients [23, 28]. Based on this premise, these studies proposed a number of enhancements to HTTP that decrease latency by reducing the number of network round trips required to transfer Web objects during a browsing session (A *Web object* consists of an HTML file along with all the files that are embedded in it by reference, and a *browsing session* is a period during which Web objects are transferred with intervening idle periods (OFF times).). We consider the effects of two of these enhancements: the use of *persistent connections* and *pipelining*. A persistent connection is a single TCP connection that is used to transfer all files during a browsing session. Pipelining allows a collection of requests to be sent to a server without waiting for a response between requests, and allows the server to respond with a minimum number of data packets. These enhancements, along with others, paved the way for the HTTP/1.1 specification which was proposed as an IETF standard in 1997 [12].

Initial evaluation of HTTP/1.1 indicated that under certain conditions it can reduce both client latency and network traffic [13]. However, the enhancements in HTTP/1.1 come at a cost. Because connections are persistent, servers must typically manage a much larger number of open connections. The work in [23] points out that requiring the server to manage many open connections could increase its processing and memory load. In general, however, a comprehensive understanding of the implications of the new features in HTTP/1.1 for servers has not been available. In particular, the effects of bottlenecks other than the network on HTTP performance have only recently begun to be studied [6].

Our study compares the impacts of using HTTP/1.0 and HTTP/1.1 when the server CPU or server disk is the bottleneck resource in the system. We also analyze in detail the network demands of each protocol. We run tests using two popular Web servers: Apache v1.3.0 running on Linux and Internet Information Server (IIS) v4.0 running on Windows NT. We use the SURGE [7] workload generator to create realistic Web workloads in a local area network environment. We analyze Web system performance by measuring throughput at the server and file latency at the client. We take detailed measurements of server subsystem (CPU, memory, disk, network) performance in order to determine where bottlenecks occur and to understand their effects. We use packet traces to analyze the network impact of our tests.

In order to use SURGE in this project, a number of enhancements to support HTTP/1.1 were added. Persistent connections and pipelining were added to the request generator per RFC 2068 [12]. The distributional model for file sizes was extended and a distributional model for the number of files requested during a browsing session was added. Details of the distributional models used in SURGE are given in Appendix A.

We exercise each server using a range of workloads that enable us to drive the server to overload and thereby expose system bottlenecks. Our results indicate that:

1. When the server is not the bottleneck, the network demands of HTTP/1.1 are somewhat lower than those of HTTP/1.0 in our LAN test configuration. Our results suggest that in a WAN environment, these differences could be significant.

2. When the server CPU is the bottleneck resource, there is no significant difference in performance between HTTP/1.0 and HTTP/1.1.

3. When memory on the server is limited and the disk system becomes the bottleneck resource, HTTP/1.1 performs significantly worse than HTTP/1.0.

The insight gained in our experiments lead us to conclude that there is a gradient along which Web performance must be considered. If a server is not heavily loaded or if the CPU or the

network is the bottleneck, the features in HTTP/1.1 are desirable due to reduction in packet traffic. However, if a server's disk system is the bottleneck, the persistent connections that HTTP/1.1 maintains between Web object transfers cause severe degradation in server throughput. Based on this observation, we propose and evaluate a connection management policy for HTTP/1.1 in which clients proactively close connections after a Web object has been transferred. This simple policy greatly improves server throughput when the disk system is the bottleneck and still reduces overall packet traffic versus HTTP/1.0.

This paper is organized as follows: Section 2 presents details on the studies that have been done to date on HTTP, Web server performance, and synthetic workload generation. Section 3 describes the experimental configuration we used to conduct tests. Section 4 presents the results of our performance analyses. Section 5 presents a summary of our findings and conclusions. We give server and SURGE configurations as appendices.

2 Related Work

2.1 Hyper Text Transfer Protocols

Presentations of the dynamics of HTTP transactions, including their interaction with TCP, can be found in [5] and [28]. HTTP/1.0 [8] is currently the most widely used version of the protocol. It requires a separate TCP connection for the transfer of each file associated with a Web object. Since busy servers can serve millions of requests per day, efficiency in setting up and tearing down connections on Web servers is important to good server performance when using HTTP/1.0 [23].

The work in [23, 28] addresses the issue of user perceived latency in Web transactions from the perspective of the network. Congestion in the Internet is a well known cause of delay in TCP transfers [29]. When network delays are assumed to be the primary source of user latency, a number of features of HTTP/1.0 become logical candidates for improvement. Specifically, HTTP/1.1 proposes a) persistent connections, b) pipelining, c) link level document compression and d) caching. We do not consider the effects of compression or caching in our study. Persistent connections reduce packet traffic since ideally only a single TCP connection is set up and used during a client browsing session on a particular server. This means that TCP connections are maintained between consecutive Web object requests (*i.e.* connections are open during OFF times), which means that many more connections can be open on a server at any moment in time. Efficiently maintaining many open connections places a different kind of burden on the server. Mogul points out in [23] that managing many open connections on the server is the biggest risk of HTTP/1.1.

HTTP/1.0 file requests are made sequentially: a request is sent to the server, the response is read by the client and then the next request is sent. Pipelining in HTTP/1.1 enables both the client and the server to avoid sending under-full packets by placing as much data (either requests or responses) as possible into each packet and placing separator strings between distinct files within a packet. Pipelining has the effect of reducing the number of request and response packets, thus reducing the number of network round trips required to complete a Web request/response transaction. Nielsen *et al.* [13] show results indicating that pipelining is necessary in order to significantly reduce latency when using HTTP/1.1.

A number of studies have analyzed the performance aspects of HTTP. In particular, [9] compares the performance of HTTP/1.0 versus HTTP/1.1 on a high latency wireless network and finds that there can be a significant benefits in HTTP/1.1. In [17], Liu and Edwards show that over one quarter of the transaction time in HTTP/1.0 is spent in TCP handshaking before any data flows.

2.2 Server Performance

Investigations of Web server performance can be divided into two categories: those which examine servers under actual use and those which examine servers using synthetic load generators. Measurements of servers under actual use are important not only for understanding how servers perform and for diagnosing their problems, but also for comparison with loads generated synthetically. Measurements of server performance using synthetic workloads suffer from the fact that they are an abstraction of actual use. However, these studies are useful for demonstrating server performance over a range of loads in a controlled fashion which is not possible in real use studies. A number of studies have characterized Web server performance using synthetically generated loads [7, 2, 5, 6, 21]. These studies have pointed out that system bottlenecks can occur in a variety of places.

A number of studies have also characterized server activity under actual use conditions [24, 16, 4, 20, 3] (in [18] there is an analysis of Web proxy logs, which are also relevant). The studies of actual use have focused primarily on server log analysis and have documented a wide variety of use characteristics. However the amount of performance data in these studies is limited. Because of the variability of real world studies and lack of reproducibility, controlled laboratory studies using synthetic workloads are necessary.

Web server performance studies using synthetic workloads have provided some key insights to system bottlenecks. In particular the work in [2] points out the overhead of lingering TCP connections in HTTP/1.0. They also show that for the workloads they consider, the server's CPU is typically the bottleneck. The work in [6] goes the furthest in exploring the details of a server under heavily loaded conditions. Through kernel profiling Banga and Mogul find inefficiencies in the *select* and *ufalloc* system calls in Digital's UNIX. Improvements in the implementations of these calls lead to significant improvements in overall performance. Their experiments were specifically designed to explore the issue of open TCP connections and their solution is at the kernel level.

2.3 Workload Generation

The task of generating workloads to test Web servers is non-trivial and has been studied intensely for a number of years [21]. Many tools have been developed which generate synthetic workloads [11, 5, 10, 27, 31, 32, 1]. All of these tools are essentially based on the WebStone model. This model makes repeated requests either at a very uniform rate, or as fast as the client system(s) can, for some set of files. At the time of writing, none of these tools were able to generate HTTP/1.1 compliant requests. Another model for workload generators are those which replay log traces [33, 26]. A hybrid of these models is the tool developed in [19]. This tool generates synthetic requests based on a statistical sampling of data from a specific server trace. While all these tools are useful for some kinds of server testing, the ideal is to measure server performance under loads which are as realistic as possible.

3 Experimental Design

The SURGE [7] workload generator is used in this project. Workloads in SURGE are based on the notion of *user equivalents*. A user equivalent (UE) is approximately the load that a single user would place on a Web server. Loads are varied in experiments using SURGE by varying the number of UE's. SURGE is based on a set of seven distinct statistical models of Web user behavior necessary to fully exercise all aspects of a Web server. These includes models for: file sizes, request sizes, document popularity, temporal locality, embedded reference count, OFF times, and session lengths.

The file size model was enhanced for this study in order to model Web objects more accurately. The session length model was added so that persistent connections could be investigated. The nature of the distributional models in SURGE are one reason why our results differ from some previously reported results. Details of the models used in SURGE and the SURGE configuration of used in our tests are listed in Appendix A.

Our experiments were conducted in an environment consisting of three PC's connected via a 100Mbps network. This was a stand alone local area network with no intervening external traffic. The PC's were configured with 200Mhz Pentium Pro CPU's with 128MB of RAM on the two SURGE client systems and between 32MB and 256MB of RAM on the server system. The SURGE client systems ran Linux 2.0.30. The server ran either Apache v1.3.0 on Linux 2.0.30, or Microsoft IIS v4.0 Windows NT Server 4.0. These servers were selected because of their popularity [25] and because they are based on a different implementation models. Apache creates a separate process for each HTTP connection, while IIS uses a lighter weight threaded model which requires less system resources for each HTTP connection. Details of the configurations for each of the servers are given in Appendices B and C.

We measured the average throughput on the server system in terms of HTTP GET operations initiated per second. On the SURGE client systems we measured the average latency for each HTTP GET. Detailed performance measurements of the servers were taken using Webmonitor (under Linux) [2], and PerfMon (under NT) with samples taken once per second during tests. When packet traces were taken, tcpdump [30] was run on the client systems.

Each experiment exercised the server over a range of loads between 20 and 640 UE's. The heavy load test levels were able to push the server into an overload state. Each experiment was run for ten minutes which was a sufficient amount of time to stabilize the subsequent measurements. SURGE was configured with 2000 unique files, resulting in about 50MB of data on the server.

Experiments were divided into three groups. The first set measures and compares the details of the network effects of each protocol using TCP packet traces. It is important to note that the development of HTTP/1.1 was based on the assumption of delays in a *wide area* network. In that environment, round trip times between clients and server can be long (hundreds of milliseconds). Our tests were run in a local area environment which means that file transfer times and connection durations tend to be much shorter on average.

The second set of experiments compares the performance of HTTP/1.0 and HTTP/1.1 when the server's CPU is the bottleneck. We increased the amount of memory on the server by powers of two until the measured performance did not change. Once this threshold size was determined, we took detailed system performance measurements using Webmonitor and Perfmon to verify that the CPU was the system bottleneck. The list of system characteristics which were tracked are given in Table 1. In these experiments we measure performance at three different load levels (low, medium and high).

The third set of experiments was designed to explore the performance differences between HTTP/1.0 and HTTP/1.1 when the disk system was the bottleneck resource on the server. To create a bottleneck at the disk, it suffices to reduce the amount of RAM available. This forces paging under heavy load since not all data files can be cached in main memory. In these experiments, we measure the latency and throughput for both Apache and IIS when memory size ranges between 32MB and 256MB. In each case the server is driven to overload.

Abbr	Webmonitor Value	Perfmon Value
USR	% CPU server processes	% CPU server processes
KER	% CPU kernel processes	% CPU kernel processes
IDL	% CPU idle process	% CPU idle process
PGR	Pages read from disk	Pages read from disk
PGW	Pages written to disk	Pages written to disk
TPS	TCP packets sent	N/A
TPR	TCP packets received	N/A
BYS	N/A	Bytes sent by server
BYR	N/A	Bytes received by server

Table 1: Detailed Performance Measurement Categories

4 Experimental Results

4.1 Network Performance

Our first set of experiments compares the network demands of HTTP/1.0 and HTTP/1.1. In order to understand the effects of both persistent connections and pipelining we capture traces of all packets transferred during tests. The network effects of persistent connections and pipelining can be expressed in terms of their impact on the quantity of each type of packet (SYN, FIN, ACK, DATA) sent during the test. Figure 1 shows the relative differences under three different client loads. The figure shows that there is a significant reduction in the number of SYN, FIN and ACK packets between HTTP/1.0 and HTTP/1.1 reflecting the benefit of persistent connections. However, the figure also shows that there is only a small change in the number of DATA packets transferred during tests. This is important because SYN, FIN and ACK packets are typically small in size when compared to DATA packets thus the total byte traffic is not greatly affected by the change in protocol. This is shown quantitatively in the first four rows of Table 2.

Evaluation of the effects of pipelining can also be seen in Table 2. The last row in the table shows that the average number of data bytes per packet remains relatively constant across loads and protocols, indicating that there is no significant impact due to pipelining. This result differs from results reported by Nielsen *et al.* [13]. This difference is due to the distributional model for Web object sizes used in our experiments. Object sizes in our study are based on an empirically derived model with a mean of 3.3 files per object and a median of 1.0 file per object. The object transferred in the Nielsen *et al.* study was composed of 43 files. We consider this size to be much larger than what is seen in typical Web objects. A similar observation is made in [17].

As we show in the next section, there is very little performance difference between HTTP/1.0 and HTTP/1.1 in the absence of a bottleneck in the disk. However, these results may be due to the use of a LAN for testing. Performance in a WAN would be strongly affected by the value of the server's congestion window[1] [15]. For this reason, we examined how the TCP congestion window (CWND) size differed between the two protocols. In the absence of congestion in the network,

[1]The TCP congestion window (CWND) is a means for throttling the flow of TCP packets in the presence of network congestion. The size of CWND begins small but grows exponentially as ACK's are received until a threshold size is reached. This mode of operation is called *slow start*. Slow start keeps the initial packet sending rate low in order to avoid saturating the network. Beyond the slow start threshold, CWND enters its *congestion avoidance* mode where it grows linearly. CWND will grow in either mode until a packet loss is detected. When a packet is lost, CWND size is dramatically reduced in order to ease network congestion.

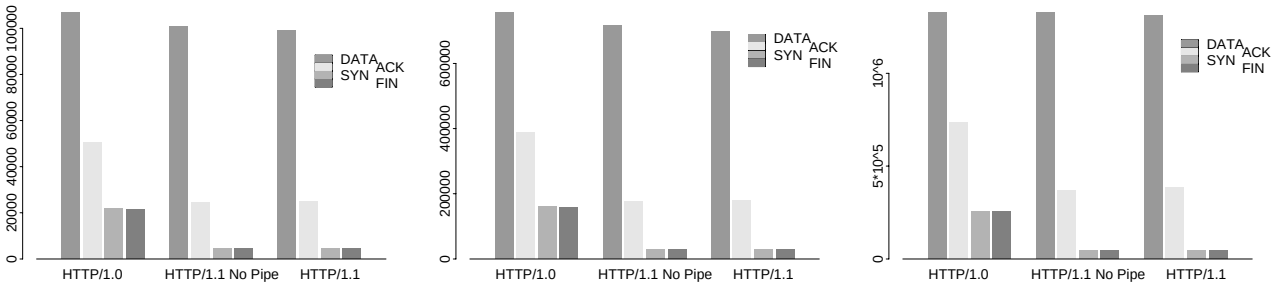

Figure 1: Counts of the total number of packets transferred of each type for 40 UE load (left) 280 UE load (center) and 520 UE load (right) in Apache v1.3.0 and 128MB

Type	HTTP/1.0			HTTP/1.1 No Pipelining			HTTP/1.1 Pipelining		
	40 UE	280 UE	520 UE	40 UE	280 UE	520 UE	40 UE	280 UE	520 UE
Data Packets	0.107	0.758	1.327	0.101	0.716	1.330	0.099	0.697	1.308
Total Packets	0.201	1.462	2.571	0.134	0.947	1.784	0.133	0.935	1.784
Total Data Bytes	128.2	897.7	1,602.9	119.0	842.8	1,607.4	118.6	836.9	1,608.7
Total Bytes	136.2	956.2	1,705.7	124.4	880.7	1,678.8	123.9	874.3	1,680.0
Data Byts/Data Pkts	1,197	1,184	1,207	1,180	1,178	1,208	1,198	1,200	1,229

Table 2: Packet and Byte Counts for Apache v1.3.0 with 128MB RAM (Counts in millions of packets or bytes)

persistent connections should enable the server to avoid TCP slow start for successive file transfers by keeping CWND large. This means that HTTP/1.1 should be more efficient in using available network bandwidth. In the presence of congestion in the network, persistent connections should more consistently keep CWND in its congestion avoidance mode. This means HTTP/1.1 should be more efficient when network bandwidth is limited as well.

We tracked the CWND size with an instrumented version of Linux 2.0.30. The modifications of Linux simply made the CWND size for each active TCP connection available for sampling. The mean (per second) and maximum CWND sizes under low, medium and high client loads are given in Table 3. The table shows that the CWND size is much larger on average in HTTP/1.1 which should have a beneficial effect on latency reduction in the wide area. In the switched 100Mbps local area environment in which the tests are run, there is no packet loss due to congestion. Thus, CWND will typically grow exponentially through the slow start threshold and then grow linearly until the connection is closed.

The distribution of CWND sizes under each protocol and load level can be seen in the cumulative distribution function (CDF) plots in Figure 2. The plots illustrate that there is relatively little difference between pipelined and non-pipelined HTTP/1.1 but a significant difference between HTTP/1.0 and HTTP/1.1. The plots show that under heavy load, nearly 80% of all HTTP/1.0 connections have a CWND size of 1 or 2 which means that most of the connections have just opened and have not yet gone through slow start. The plots also show that over 50% of the HTTP/1.1 have a CWND size above 7 indicating that in a WAN, much higher average transfer rates should be achievable under HTTP/1.1.

	HTTP/1.0			HTTP/1.1 No Pipelining			HTTP/1.1 Pipelining		
	40 UE	280 UE	520 UE	40 UE	280 UE	520 UE	40 UE	280 UE	520 UE
CWND Average	5.79	4.04	3.81	10.48	10.79	9.28	11.04	11.30	9.72
CWND Max	29	83	99	68	186	150	106	202	141

Table 3: Congestion window comparison for Apache v1.3.0 with 128MB RAM

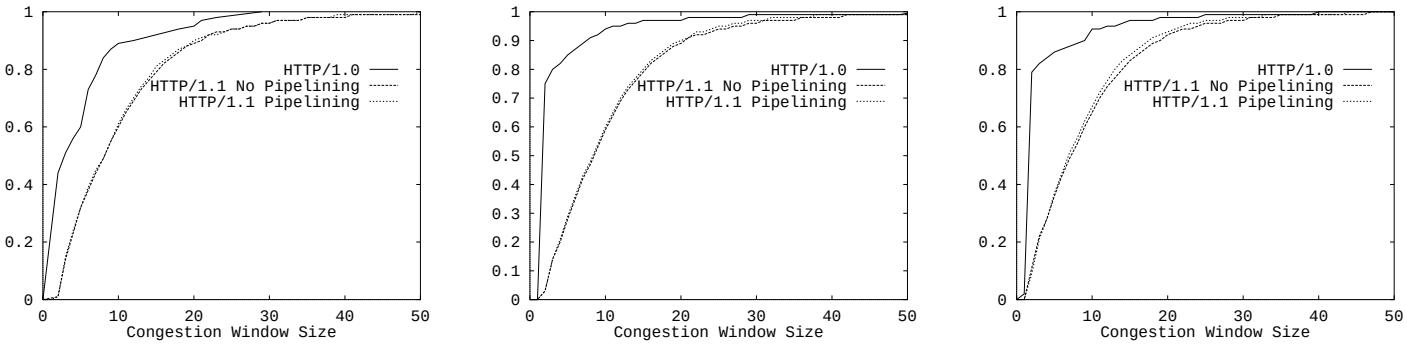

Figure 2: Cumulative Distribution Function plots of congestion window sizes for 40 UE load (left) 280 UE load (center) and 520 UE load (right) in Apache v1.3.0 and 128MB

4.2 CPU Constrained Performance

In this section we examine the case in which the CPU is the bottleneck in the system's performance. We ran performance tests with the server configured with sufficient RAM to contain the Web server, operating system and all data files in main memory. Our test file set consists of approximately 50MB of data; Apache on Linux 2.0.30 uses approximately 14MB while IIS on NT 4.0 uses approximately 30MB. The server for this set of experiments was configured with 128MB of memory.

Performance monitoring data gives us better insight into the details of system utilization in this configuration. The performance monitors enable us to track the low level system components on the servers that are listed in Table 1. The results of the detailed system performance measurements for Apache and IIS are given in Tables 4 and 5 respectively. Each test was run with a warm file cache, meaning that each test was run twice and measurements were taken during the second test.

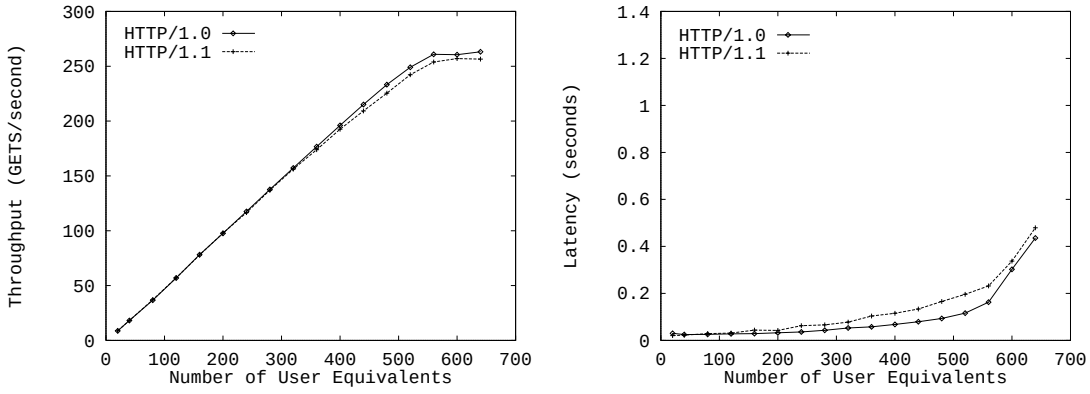

Figure 3: Throughput (left) and latency (right) for Apache v1.3.0 with 128MB memory

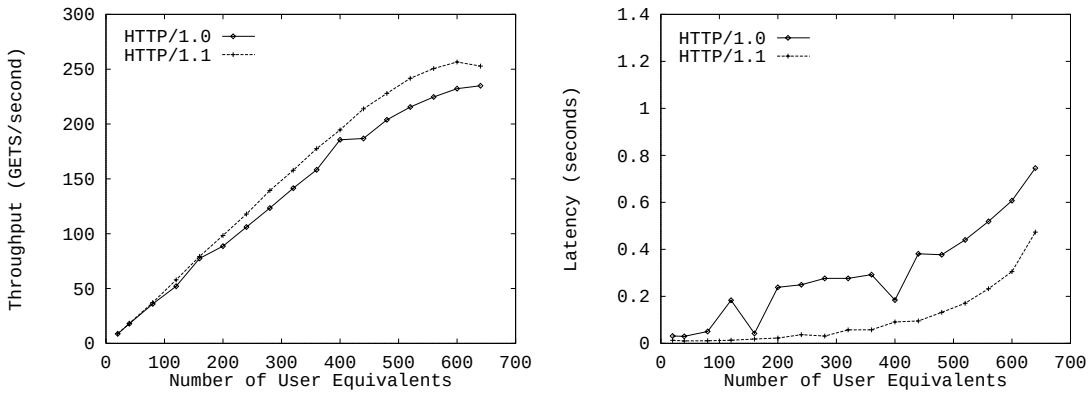

Figure 4: Throughput (left) and latency (right) for IIS v4.0 with 128MB memory

Value	HTTP/1.0			HTTP/1.1		
	40 UE	280 UE	520 UE	40 UE	280 UE	520 UE
USR	1.66	15.18	27.15	1.33	13.10	20.84
KER	7.89	41.95	69.96	6.90	43.70	72.02
IDL	90.45	42.87	2.89	91.77	43.20	7.14
PGR	0.00	0.00	0.01	0.02	0.02	0.02
PGW	11.16	33.74	48.93	12.44	36.73	46.38
TPS	233	1,859	3,515	188	1,382	2,135
TPR	125	944	1,745	66	441	710

Table 4: Detailed Performance Measurements for Apache v1.3.0 with 128MB RAM

This gives a clearer picture of the steady state operation of the server under each of the loads. The disk activity data shows that there is virtually no paging taking place during tests (except for writing, which is due to server log file generation) indicating that the disk system is not the bottleneck during tests. The network activity data shows a mean transfer rate of approximately 22Mbps for Apache and 25Mbps for IIS. These are well below the 100Mbps capacity of the network indicating that the network is not the bottleneck during tests. Furthermore, the very low CPU percent idle times indicate that the CPU is the limiting resource under the 520 UE load.

Performance results are shown over a range of client loads in Figure 3 for Apache and Figure 4 for IIS. The graphs indicate that for the Apache server, performance does not vary greatly between HTTP/1.0 and HTTP/1.1, and that for the IIS server, performance under HTTP/1.1 is only slightly better than under HTTP/1.0. These results show that the new features added to HTTP/1.1 do not significantly reduce the server's CPU processing load per byte transferred.

4.3 Disk Constrained Performance

As the amount of RAM on the server is reduced to a level where paging of data files becomes necessary, the disk system becomes the bottleneck in the system. Performance results are shown over a range of memory configurations and client loads, in Figure 5 for Apache and HTTP/1.0, in Figure 6 for Apache and HTTP/1.1, in Figure 7 for IIS and HTTP/1.0 and in Figure 8 for IIS and HTTP/1.1. The graphs show that performance degrades dramatically when memory on the server is limited. While this by itself may not be surprising, what is particularly interesting is that the

Value	HTTP/1.0			HTTP/1.1		
	40 UE	280 UE	520 UE	40 UE	280 UE	520 UE
USR	1.07	4.70	7.32	1.10	4.61	7.59
KER	6.77	46.45	78.26	6.46	46.85	80.92
IDL	92.16	48.85	14.42	92.44	48.54	11.49
PGR	0.00	0.00	0.00	0.00	0.00	0.00
PGW	0.00	0.00	0.00	0.00	0.00	0.00
BYS	232,100	1,829,474	2,967,467	230,295	1,840,909	3,177,293
BYR	1,045	7,832	12,332	1,362	10,149	16,769

Table 5: Detailed Performance Measurements for IIS/NT with 128MB RAM

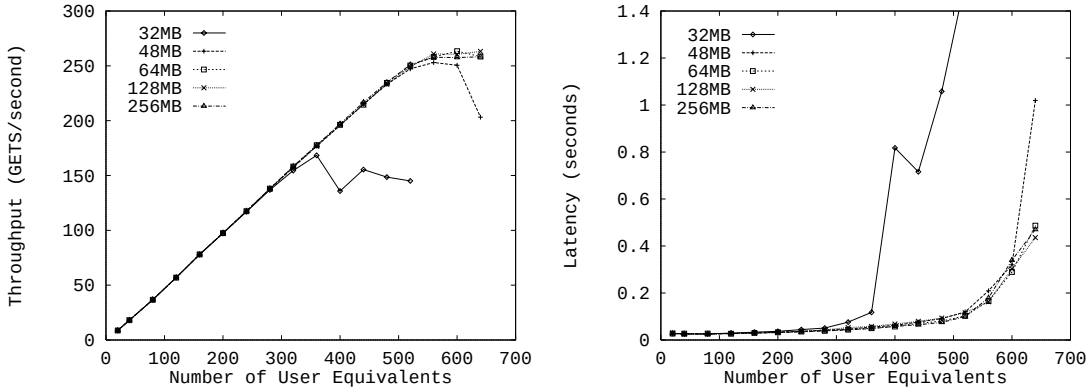

Figure 5: Throughput (left) and latency (right) for Apache v1.3.0 and HTTP/1.0

degradation of performance is much more pronounced under HTTP/1.1 than under HTTP/1.0.

4.3.1 Memory Use and TCP Connections

The differences in performance between HTTP/1.0 and HTTP/1.1 when the disk system is the constraining resource can be understood through more detailed measurements of memory use. We measured the amount of memory used and the number of TCP connections in the ESTABLISHED state[1] for HTTP/1.0 and HTTP/1.1 on Apache under a range of low loads. (We attempted to take the same measurement for IIS however, due to NT's methods of memory allocation, this was not possible.) We fit a least squares line to the points to determine the per connection memory use. The results on the left side of Figure 9 show the per connection memory cost under HTTP/1.0 to be approximately 378KB, while the plot on the right side of of Figure 9 shows the per connection memory cost under HTTP/1.1 to be about 81KB. These figures indicate that the memory needed by a connection in Apache is large; for an active connection (*i.e.* an HTTP/1.0 connection) the memory demand is four times higher than for a mostly-idle connection (*i.e.* an HTTP/1.1 connection).

These results are significant because when main memory is full, creating a new HTTP connection will require that data files be paged from memory; subsequent requests for these files will no longer hit in the cache and will require disk I/O. The mean file size for our data set was 26KB, and the

[1] A TCP connection is in the ESTABLISHED state after the three way packet exchange which sets up a connection has been completed and before a connection has been closed. Data can be transferred in either direction when a connection is in the ESTABLISHED state.

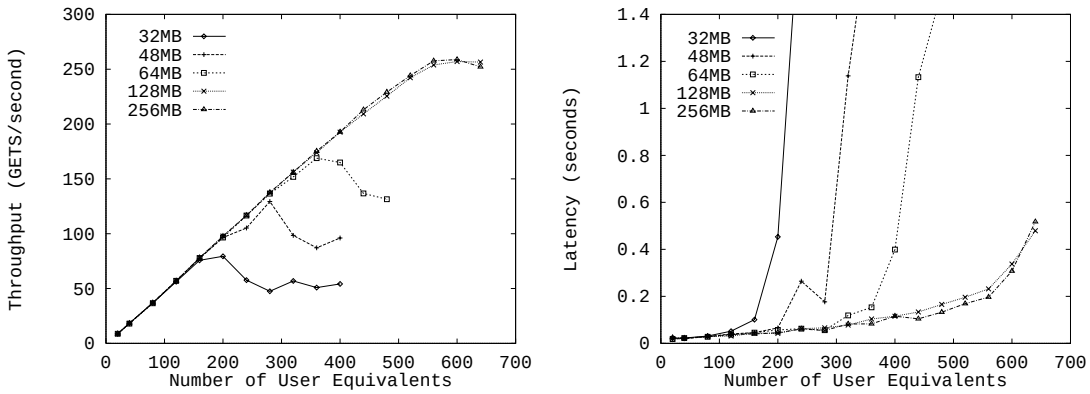

Figure 6: Throughput (left) and latency (right) for Apache v1.3.0 and HTTP/1.1

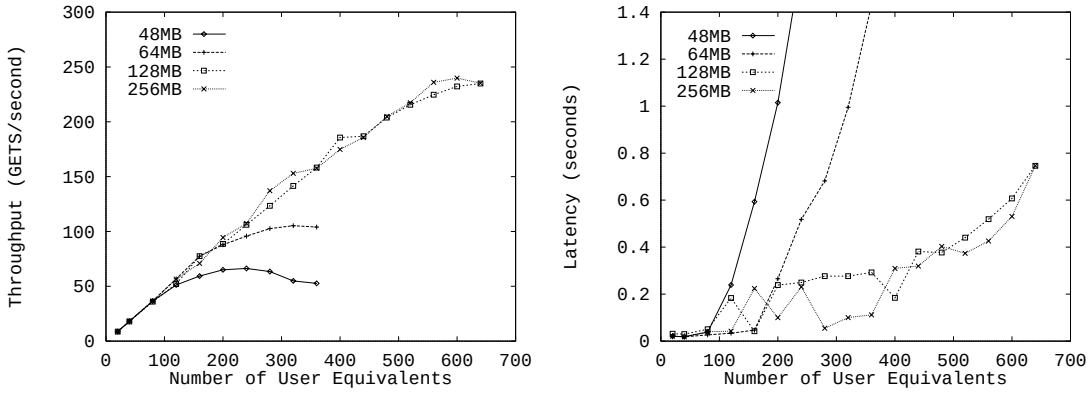

Figure 7: Throughput (left) and latency (right) for IIS v4.0 and HTTP/1.0

median was 3KB. In the case of HTTP/1.1, this means that when memory is full, about 3 files on average must be paged out of memory in order to accommodate a new connection — and in the common case (files of median size or less), 25 files or more may be evicted in order to create a new connection.

While these results show that Apache uses a large amount of memory per connection, obtaining corresponding results for the IIS server is more difficult (because precise memory allocation measurements in Windows NT are hard to obtain from the Perfmon tool). However, our initial investigations indicate that IIS uses less memory per connection, perhaps because of its thread-based implementation. This helps to explain why the performance degradation under HTTP/1.1 as compared to HTTP/1.0 is less pronounced in the IIS server (Figures 7 and 8).

Since connection effects are so important, we measured the number of TCP connections made under each protocol. Table 6 shows these measurements under three different load levels. The second line in the table shows the mean number of TCP connections in the ESTABLISHED state. This shows that there are many more TCP connections made in HTTP/1.0 but that they are very short lived on average. The third line in the table, average connection duration, highlights the effect of persistent connections. Connection duration for HTTP/1.0 is simply the average file transfer latency, but connection duration for HTTP/1.1 includes both file transfer latency and idle ("OFF") periods. The table shows that connection duration in HTTP/1.1 is dominated by the OFF periods. The last line in the table is the product of lines one and three. It is the total number of connection seconds during the test. The connection seconds for HTTP/1.0 represent demands

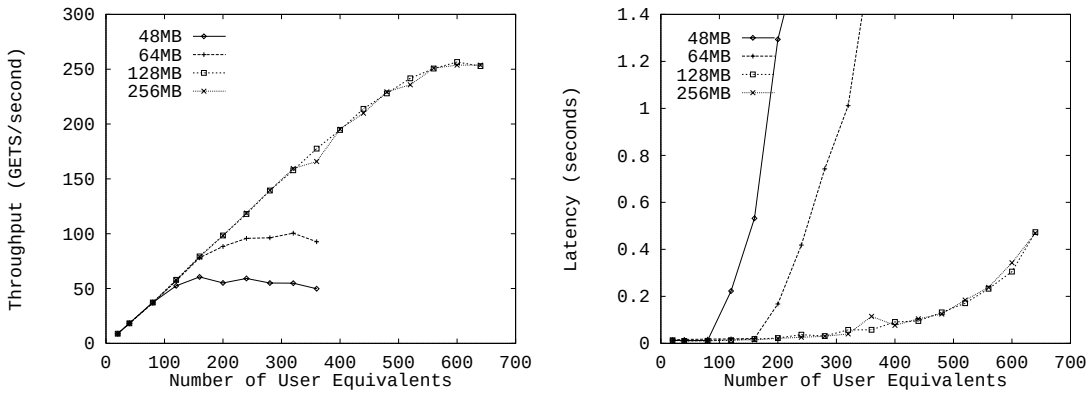

Figure 8: Throughput (left) and latency (right) for IIS v4.0 and HTTP/1.1

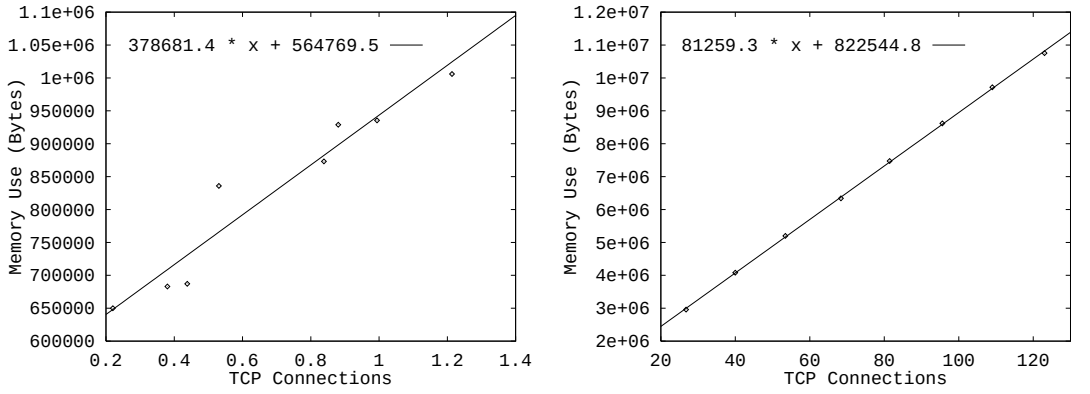

Figure 9: Memory use per TCP connection for HTTP/1.0 (left) and HTTP/1.1 (right) for Apache v1.3.0 and 128MB

to transfer data, while this value for HTTP/1.1 includes open but idle connections. The ratio of these two values shows that for high loads, HTTP/1.1 can place more than *20 times* the connection load on the server that HTTP/1.0 does.

The per connection cost can also explored by configuring SURGE clients to open multiple connections per UE. In this experiment, two connections per UE are used by SURGE clients to transfer files when a Web object has more than one embedded file. Figure 10 shows the impact on throughput for both Apache and IIS when multiple connections are used under HTTP/1.1. In both cases throughput is decreased. Not only does this highlight the cost per connection issue, but it also suggests that multiple connections per client used as a default setting in some popular browsers may have a serious negative impact on server throughput.

4.4 HTTP/1.1 Connection Management

Analysis of the low utilization of most connections under HTTP/1.1 leads us to conclude that when a server is constrained by its disk system, the relative benefit of keeping connections open across successive Web object transfers is much smaller than the benefit of keeping the connection open during the transfer of a single object. Thus, an effective policy for managing connections may be to keep the persistent, pipelined connection open for the transfer of a Web object but to close the connection after each web object transfer. This enhancement is similar to the GETALL construct proposed in [28]. GETALL is a request sent by the client in which the server responds by sending

	HTTP/1.0			HTTP/1.1		
	40 UE	280 UE	520 UE	40 UE	280 UE	520 UE
Total Connects	10,902	83,826	153,231	2,278	15,194	26,456
Average ESTABLISHED Connections	0.16	3.47	23.82	35.41	234.11	398.05
Average Connection Duration (sec)	0.02	0.04	0.12	13.02	13.59	15.59
Connect Seconds	218	3,353	18,388	29,659	206,486	412,449

Table 6: TCP Connection Measurements for Apache v1.3.0 with 128MB RAM

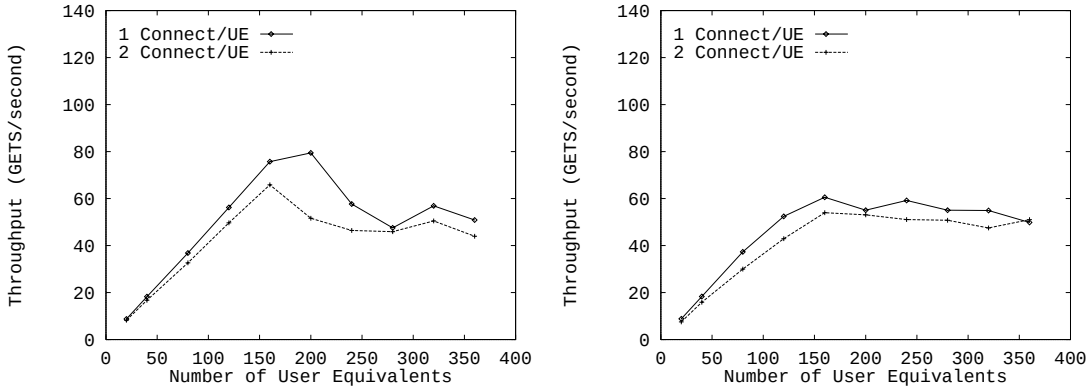

Figure 10: Effects of multiple connections for Apache v1.3.0 with 32MB (left) and IIS v4.0 with 48MB (right)

an entire Web object. Our policy simply closes the connection after each GETALL. We call this policy *early close*.

If we consider that there are F files per object and N objects requested during a browsing session to a single server, then the effects of each protocol are as follows:

- HTTP/1.0 will make $F * N$ connections per client which are only open during the transfer of each file,
- HTTP/1.1 with early close will make N connections per client which are only open during the transfer of each Web object,
- Typical HTTP/1.1 will maintain 1 connection per client (note: servers are typically configured with a timeout on persistent connections, thus 1 connection per client is the best possible case), which is open during Web object transfers and OFF times.

We expect that HTTP/1.1 with early close should perform somewhere between HTTP/1.0 and standard HTTP/1.1 in terms of server throughput and network efficiency. The performance results of HTTP/1.1 with early close can be seen in Figures 11 and 12 for Apache under 32MB and 128MB memory configurations. The figures show that HTTP/1.1 with early close actually performs as well as HTTP/1.0 in terms of throughput in either configuration and much better than standard HTTP/1.1 in the disk constrained test.

The drawback of HTTP/1.1 with early close is that closing connections between Web object transfers will increase network traffic due to the extra connection startups. Details of the impact of HTTP/1.1 with early close on the network are given in Table 7. From this we can see that

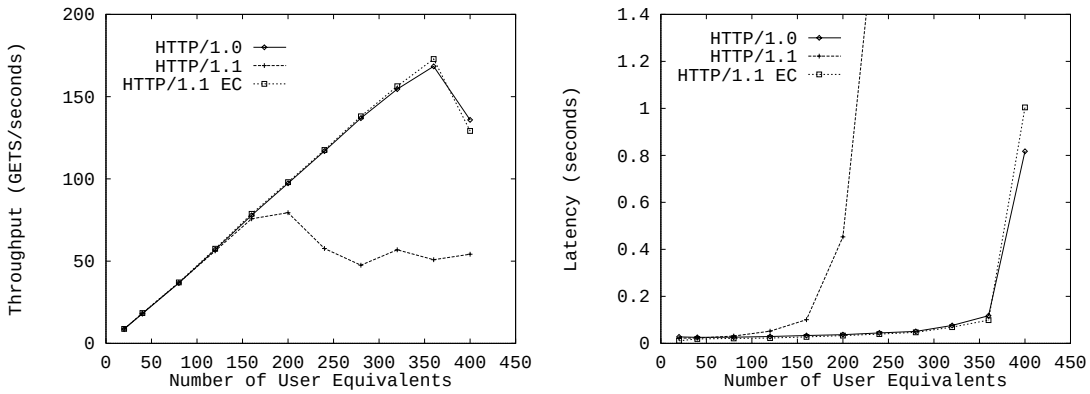

Figure 11: Throughput (left) and latency (right) for Apache v1.3.0 with 32MB memory and HTTP/1.1 with early close (EC)

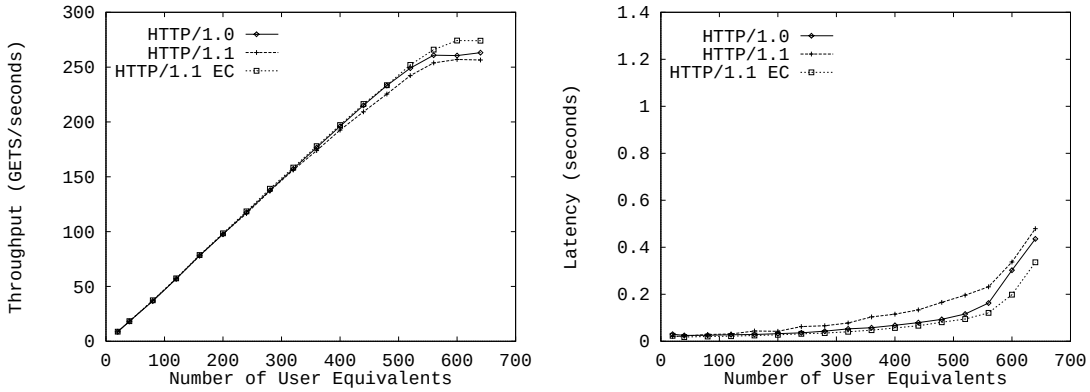

Figure 12: Throughput (left) and latency (right) for Apache v1.3.0 with 128MB memory and HTTP/1.1 with early close (EC)

HTTP/1.1 with early close roughly falls in between HTTP/1.0 and standard HTTP/1.1 in terms of total connections opened during tests and that only 17% more packets are transferred versus standard HTTP/1.1. Details of the counts for each type of packet are compared in Figure 13. This figure again shows that HTTP/1.1 with early close falls roughly between HTTP/1.0 and standard HTTP/1.1 as expected.

5 Conclusions

In this paper we have measured and compared the relative performance impact of HTTP/1.0 and HTTP/1.1 in terms of both transfer latency and file throughput for two popular Web servers. We conducted our experiments in a LAN using the SURGE workload generator. We measured the network demands of each protocol by taking packet traces during tests, and used detailed performance monitoring to understand the the server demands.

Our study is limited to performance effects that occur in a LAN in which round trip times are short and packet loss is rare. For that reason we do not observe some performance improvements of HTTP/1.1 that would occur in a wide area network setting. However, we do measure quantities such as the server congestion window that help us infer how HTTP/1.1 performs in a wide area setting.

Type	HTTP/1.0			HTTP/1.1 Pipelining			HTTP/1.1 early close		
	40 UE	280 UE	520 UE	40 UE	280 UE	520 UE	40 UE	280 UE	520 UE
Total Connects	10,902	83,826	153,231	2,278	15,194	26,456	7,876	54,667	96,776
CWND Average	5.79	4.04	3.81	11.04	11.30	9.72	4.87	5.40	4.39
CWND Max	29	83	99	106	202	141	35	233	182
Data Packets	0.107	0.758	1.327	0.099	0.697	1.309	0.105	0.703	1.220
Total Packets	0.201	1.462	2.571	0.133	0.935	1.784	0.176	1.180	2.089
Total Data Bytes	128.2	897.7	1,602.9	118.6	836.9	1,608.7	127.3	844.4	1,493.6
Total Bytes	136.2	956.2	1,705.7	123.9	874.3	1,680.0	134.3	891.6	1,577.1
Data Byts/Data Pkt	1,197	1,184	1,207	1,198	1,200	1,229	1,210	1,200	1,224

Table 7: Network impact and Packet/Byte Counts (in millions) for HTTP/1.1 with early close on Apache v1.3.0 with 128MB RAM

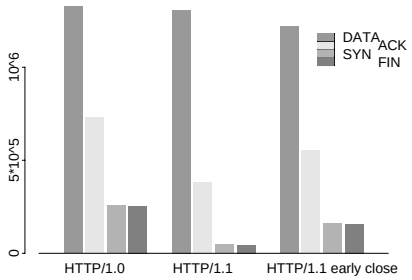

Figure 13: Counts of the total number of packets transferred of each type for 520 UE load in Apache v1.3.0 and 128MB using HTTP/1.1 with early close

Our measurements confirm and quantify the degree to which HTTP/1.1 reduces network load compared to HTTP/1.0. We show that the persistent connection feature of HTTP/1.1 lowers the number of SYN, FIN and ACK packets transferred significantly. Our measurements of server congestion window size indicate that HTTP/1.1 should enable more efficient utilization of network bandwidth in the wide area. On the other hand, our measurements of the number of data packets transferred and the number of data bytes per packet indicate that the pipelining feature of HTTP/1.1 has very little impact on the network. Our measurements indicate that pipelining does not significantly reduce file transfer latency which differs from previously reported results in [13]. This difference can be traced to the shorter round trip times in a LAN environment and the smaller Web object size used in our study.

We show that there is relatively little difference in performance between HTTP/1.0 and HTTP/1.1 when the CPU is the bottleneck in the system. These results show that the new features added to HTTP/1.1 do not significantly reduce the server's CPU processing load per byte transferred.

We conducted experiments in which we forced the disk system to become the bottleneck. We find that when there is insufficient RAM to accommodate the *entire* data file set, that performance using HTTP/1.1 can be much worse than HTTP/1.0. This is because when using persistent connections, there are many more open connections on average on the server. We show that per connection memory cost can be as high as 81KB per connection on Apache v1.3.0 which means that paging is more likely to occur in HTTP/1.1 when memory is limited. We show that multiple connections

per client causes server throughput to degrade compared to a single connection per client.

Our observation of the underutilization of persistent connections in HTTP/1.1 leads us to propose a connection management policy. This policy, which we call early close, closes connections at the end of a Web object transfer. The intention of early close is to increase server throughput, while maintaining the beneficial network effects of persistent connections, when the disk system is the bottleneck. We measure the effect of the early close policy by having clients close their HTTP connections after the transfer of each Web object. Our measurements show that early close generates approximately 17% more network packet traffic than standard HTTP/1.1. However, our measurements show that when the disk system is the bottleneck, server throughput is greatly increased by using early close.

Acknowledgements The authors of this paper would like to thank Robert Frangioso for help with the set-up and configuration of the Microsoft NT/IIS environment. The authors would like to thank David Martin for his help with Linux, Webmonitor and Apache. The authors would also like to thank Virgílio Almeida for making Webmonitor available to us.

References

[1] WebBench 2.0. http://www.zdnet.com/zdbop/webbench/webbench.html.

[2] Jussara Almeida, Virgilio Almeida, and David Yates. Measuring the behavior of a world wide web server. In *Proceedings of the Seventh IFIP Conference on High Performance Networking (HPN)*, White Plains, NY, April 1997.

[3] Virgilio Almeida, Azer Bestavros, Mark Crovella, and Adriana de Oliveira. Characterizing reference locality in the WWW. In *Proceedings of 1996 International Conference on Parallel and Distributed Information Systems (PDIS '96)*, pages 92–103, December 1996.

[4] Martin Arlitt and Cary Williamson. Web server workload characterization: The search for invariants. In *Proceeding of the ACM SIGMETRICS '96 Conference*, Philadelphia, PA, April 1996.

[5] Gaurav Banga and Peter Druschel. Measuring the capacity of a web server. In *Proceedings of the USENIX Annual Technical Conference*, Monterey, CA, December 1997.

[6] Gaurav Banga and Jeffrey Mogul. Scalable kernel performance for internet servers under realistic loads. In *Proceedings of the USENIX Annual Technical Conference*, New Orleans, LA, June 1998.

[7] Paul Barford and Mark Crovella. Generating representative workloads for network and server performance evaluation. In *Proceedings of ACM SIGMETRICS '98*, pages 151–160, Madison, WI, June 1998.

[8] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext transfer protocol – HTTP/1.0. IETF RFC 1945, October 1995.

[9] Stephen Cheng, Kevin Lai, and Mary Baker. Analysis of HTTP/1.1 on a Wireless Network. Technical report, Stanford University, 1998.

[10] HTTP Client. http://www.innovation.ch/java/HTTPClient/.

[11] The Standard Performance Evaluation Corporation. Specweb96. http://www.specbench.org/org/web96/, 1997.

[12] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext transfer protocol – HTTP/1.1. IETF RFC 2068, January 1997.

[13] H. Frystyk-Nielsen, J. Gettys, A. Baird-Smith, E. Prud'hommeaux, H. Wium-Lie, and C. Lilley. Network performance effects of HTTP/1.1, CSS1 and PNG. In *Proceedings of ACM SIGCOMM '97*, Cannes, France, September 1997.

[14] Bernardo Huberman, Peter Pirolli, James Pitkow, and Rajan Lukose. Strong regularities in world wide web surging. *Science*, 280:95–97, 1998.

[15] Van Jacobson. Congestion avoidance and control. In *Proceedings of ACM SIGCOMM '88*, pages 314–329, August 1988.

[16] T.T. Kwan, R.E. McGrath, and D.A. Reed. User access patterns to NCSA's WWW server. Technical Report UIUCDCS-R-95-1934, University of Illinois, Department of Computer Science, February 1995.

[17] Binzhang Liu and Edward Fox. Web traffic latency: Characteristics and implications. In *WebNet98*, Orlando, FL, November 1998.

[18] C. Maltzahn, K. Richardson, and D. Grunwald. Performance issues of enterprise level web proxies. In *Proceedings of ACM SIGMETRICS '97*, Seattle, WA, June 1997.

[19] S. Manley, M.Courage, and M. Seltzer. A self-scaling and self-configuring benchmark for web servers. Harvard University, 1997.

[20] Stephen Manley and Margo Seltzer. Web facts and fantasy. In *Proceedings of the 1997 USENIX Symposium on Internet Technologies and Systems*, Monterey, CA, December 1997.

[21] Robert McGrath. Measuring the performance of http daemons. http://www.ncsa.uiuc.edu/InformationServers/Performance/Benchmarking/bench.html, 1996.

[22] Mindcraft. WebServer Comparison: Microsoft Windows NT Server 4.0 on a Compaq ProLiant 3000 and Sun Solaris 2.6 with Sun WebServer 1.0 on a Sun Ultra Enterprise 450. http://www.mindcraft.com/whitepapers/.

[23] Jeffrey Mogul. The case for persistent-connection HTTP. Technical Report WRL 95/4, DEC Western Research Laboratory, Palo Alto, CA, 1995.

[24] Jeffrey Mogul. Network behavior of a busy web server and its clients. Technical Report WRL 95/5, DEC Western Research Laboratory, Palo Alto, CA, 1995.

[25] Netcraft. Netcraft web server survey. http://www.netcraft.co.uk/, August 1998.

[26] Henrik Frystyk Nielsen. Libwww. http://www.w3.org/Library/.

[27] University of Minnesota. Gstone version 1. http://web66.coled.umn.edu/gstone/info.html.

[28] Venkat Padmanabhan and Jeffrey Mogul. Improving HTTP latency. *Computer Networks and ISDN Systems*, 28:25–35, December 1995.

[29] Vern Paxson. End-to-end internet packet dynamics. In *Proceedings of ACM SIGCOMM '97*, Cannes, France, September 1997.

[30] tcpdump. http://ftp.ee.lbl.gov/tcpdump.tar.Z.

[31] Gene Trent and Mark Sake. Webstone: The first generation in http server benchmarking, February 1995. Silicon Graphics White Paper.

[32] WebCompare. http://webcompare.iworld.com/.

[33] Webjamma. http://www.cs.vt.edu/~chitra/webjamma.html.

A Surge Models and Configuration

The SURGE workload generator in this project was an updated version of the tool. Persistent connections and pipelining were added to the request generator in order to support HTTP/1.1. The distributional model for file sizes was extended in order to more accurately represent Web objects. A distributional model for the number of files requested during a browsing session was added as well. SURGE contains models for each of the following characteristics of Web use:

- **File sizes.** File sizes refer to the sizes of files which make up the test set on the Web server. Accurate modeling of this distribution is necessary in order to exercise the server's file system in a representative manner. File sizes are divided into three categories in order to more accurately represent Web objects. Base files refer to HTML files which contain embedded files. Embedded files refer to files which are referenced by base files. Single files are files which are neither base nor embedded. Both base files and single files are modeled with hybrid distributions.
- **Request sizes.** Request sizes are the sizes of files which are transferred by SURGE during tests. Accurate modeling of this distribution is required in order to exercise the network components of the system in a representative manner. Request sizes are modeled with a hybrid distribution.
- **Document popularity.** Popularity is the distribution of requests on a per-file basis. Accurate modeling of this distribution is required in order to properly exercise the caches and buffers in the system.
- **Temporal locality.** Temporal locality refers to the likelihood that, once a file has been requested, that it will be requested again in the near future. Accurate modeling of temporal locality is required in order to properly exercise caches and buffers in the system.

Component	Model	Probability Density Function	Parameters
File Sizes – Base File Body	Lognormal	$p(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-(\ln x-\mu)^2/2\sigma^2}$	$\mu = 7.630$; $\sigma = 1.001$
File Sizes – Base File Tail	Pareto	$p(x) = \alpha k^\alpha x^{-(\alpha+1)}$	$k = 10K$; $\alpha = 1.0$
File Sizes – Embedded File	Lognormal	$p(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-(\ln x-\mu)^2/2\sigma^2}$	$\mu = 8.215$; $\sigma = 1.460$
File Sizes – Single File1	Lognormal	$p(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-(\ln x-\mu)^2/2\sigma^2}$	$\mu = 7.101$; $\sigma = 1.200$
File Sizes – Single File2	Lognormal	$p(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-(\ln x-\mu)^2/2\sigma^2}$	$\mu = 11.151$; $\sigma = 1.143$
Request Sizes – Body	Lognormal	$p(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-(\ln x-\mu)^2/2\sigma^2}$	$\mu = 7.881$; $\sigma = 1.339$
Request Sizes – Tail	Pareto	$p(x) = \alpha k^\alpha x^{-(\alpha+1)}$	$k = 34102$; $\alpha = 1.177$
Popularity	Zipf		
Temporal Locality	Lognormal	$p(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-(\ln x-\mu)^2/2\sigma^2}$	$\mu = 1.5$; $\sigma = 0.80$
OFF Times	Pareto	$p(x) = \alpha k^\alpha x^{-(\alpha+1)}$	$k = 1$; $\alpha = 1.4$
Embedded References	Pareto	$p(x) = \alpha k^\alpha x^{-(\alpha+1)}$	$k = 2$; $\alpha = 1.245$
Session Length	Inverse Gaussian	$p(x) = \sqrt{\lambda/2x^3\pi} e^{-\lambda(x-\mu)^2/2x\mu^2}$	$\mu = 3.86$; $\lambda = 6.08$

Table 8: Summary of SURGE Configuration Parameters used in tests

- **Embedded reference count.** Embedded reference count is the number of files plus the base HTML file that make up a Web object. Accurate modeling of this distribution is important, especially in HTTP/1.1, since the benefits of pipelining are chiefly determined by this value.
- **Off times.** OFF times are the user "think" times between successive Web object requests. Accurate modeling of this distribution is necessary both to capture the bursty nature of Web requests and to determine the length of time persistent connections are held open in HTTP/1.1.
- **Session lengths.** Session lengths refer to the number of Web objects a user will request from a particular Web server before moving onto another Web server (*i.e.* during a browsing session). Accurate modeling of this distribution is important in HTTP/1.1 tests since it determines the number of Web objects which will be requested for the duration of a persistent connection. This distribution is a new addition to SURGE for the purpose of testing HTTP/1.1. Data for this characteristic is based on the work in [14].

Each of these models is combined in SURGE in order to generate a representative request stream at the server. A summary of the model distributions and parameters used in SURGE is given in Table 8.

B IIS/NT Configuration Parameters

IIS was configured to be as efficient as possible for the tests in this study. The configuration used was published in [22] and is given in Table 9.

C Apache Configuration Parameters

Apache v1.3.0 was configured to be as efficient as possible for the tests in this study. The configuration used was based on the highperformance.conf file which is supplied with the distribution. The specific parameters used are given in Table 10.

Feature	Configuration
Operating System	Windows NT Server v4.0; Service Pack 3 NT; Windows NT Option Pack updates: w3svc (vesion 4.02.0636) 12/97 TCP/IP and AFD Transport hotfix IIS 4.0 logfix 4/98 NDIS hotfix 12/97
Tuning	Performance set to maximize file sharing; foreground application boost set to NONE; ran chkdsk /l:65536 on disk to increase the log size. The following registry settings in HKEY_LOCAL_MACHINE,SYSTEM,CurrentControlSet,Services: Tcpip,Parameters,MaxFreeTcbs 72000 Tcpip,Parameters,MaxHashTableSize 65536 Tcpip,Parameters,TcpTimedWaitDelay 60 Tcpip,Parameters,TcpWindowSize 17520
Web Server	Internet Information Server 4.0
Tuning	Performance set to handle over 100,000 hits per day; removed all mappings except asp., connection close set to 15 seconds The following registry settings in HKEY_LOCAL_MACHINE,SYSTEM,CurrentControlSet,Services: InetInfo,Parameters,ListenBackLog 250 InetInfo,Parameters,MaxPoolThreads 2 InetInfo,Parameters,MemoryCacheSize 10000000 InetInfo,Parameters,ObjectCacheTTL 0xffffffff InetInfo,Parameters,OpenFileInCache 7000 InetInfo,Parameters,PoolThreadLimit 2

Table 9: Windows NT 4.0/IIS v4.0 Configuration Parameters

Parameter	Value
KeepAlive	On
MaxKeepAliveRequests	150
KeepAliveTimeout	15
MinSpareServers	5
MaxSpareServers	10
StartServers	5
MaxClients	900
MaxRequestsPerChild	10000000

Table 10: Apache v1.3.0 Configuration Parameters