

Web Server Performance in a WAN Environment

Vsevolod V. Panteleenko and Vincent W. Freeh

TR-02-02

Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN 46556
{vvp, vin}@cse.nd.edu

Abstract—This work analyzes web server performance under simulated WAN conditions. The workload simulates many critical network characteristics, such as network delay, bandwidth limit for client connections, and small MTU sizes to dial-up clients. A novel aspect of this study is the examination of the internal behavior of the web server at the network protocol stack and the device driver. Many known server design optimizations for performance improvement were evaluated in this simulated environment.

We discovered that WAN network characteristics may significantly change the behavior of the web server compared to the LAN-based simulations and make many of optimizations of the server design irrelevant in this environment. Particularly, we found out that small MTU size of the dial-up user connections can increase the processing overhead several times. At the same time, the network delay, connection bandwidth limit, and usage of HTTP/1.1 persistent connections do not have a significant effect on the server performance. We have found there is little benefit due to copy and checksum avoidance, optimization of request concurrency management, and connection open/close avoidance under a workload with small MTU sizes, which is common for dial-up users.

I. INTRODUCTION

Web server performance, which is crucial to large public web sites, has been extensively studied in the literature [5][7][16][29]. These studies are used for web site performance evaluation and capacity planning as well as for development of techniques for web server performance improvement [8][12][19]. Many of these studies use simulated workload in local area network (LAN) environment. However, web servers deployed in the real world often show significantly different behavior from these studies [23][26][35]. The difference can be explained by failure to reproduce some of the network characteristics of the real world load. Public web servers operate in a wide area network (WAN) environment with

network characteristics quite different from a LAN, such as by high network delay, limited bandwidth to a large portion of the dial-up clients, small packet (MTU) size of connections, and packet losses. Therefore, even though web server performance has been extensively studied, there is need for more study.

In a recent paper, Nahum et. al. [29] consider the effect of more realistic network characteristics and workloads on web servers, particularly effect of network delay. In contrast to their work, which mainly looks at the server behavior visible to a client, this paper studies the internal behavior of the web server at the network protocol stack and the device driver. It also examines client characteristics, such as latency and maximum sustained request rate, using realistic workloads. These workloads emulate a wide range of real-life network characteristics, such as network delay, limited bandwidth of connections, small MTU sizes and others. This study also evaluates known techniques for server performance improvement under WAN conditions. We used two web servers for the experiments: Apache [4] and TUX [38], both running on Linux operating system.

In our experiments we discovered the following.

- Similar to experiments in LAN environment [16], most of the processing overhead for a web server under WAN conditions is due to processing in network protocol stack and device driver.
- Reduced MTU size of the client connections can increase server processing overhead several times.
- Network delay, connection bandwidth limit, and using HTTP/1.1 persistent connections does not have significant effect on the server performance.
- End-user request latency mostly depends on the bandwidth limit of the connections and to some extent on the network delay.

Additionally, this paper disputes claims made in previous studies regarding the benefit of web server optimizations. In particular, we present results that indicate, under WAN conditions with small MTU size, there is little benefit to (i) copy and checksum avoidance [19][30], (ii) optimizations of request concurrency management by optimizing *select* system call [8][12] or performing computations at the socket event handler [19], and (iii) connection open/close avoidance using HTTP/1.1 persistent connections [21][27]. This is contrary to claims made previously based on measurements in LAN environment.

It was necessary to create or modify several tools to collect data for this paper. First, we created a tool that emulates network delay and bandwidth limitations of client connections. Second, the SURGE workload generator [9] was modified to eliminate server throttling [5]. Lastly, the kernel was instrumented in order to provide profiling data for determining where request processing occurs.

The paper is organized as follows. The next section describes related work. Section III summarizes known optimizations of the web server design in order to increase the server performance. Section IV describes our test methodology and the experimental setup. Section V gives the results for the measurements of the server performance and evaluates the efficiency of the optimization techniques described in Section III. Finally, Section VI concludes.

II. RELATED WORK

There are several standard workload generation tools used for web server performance analysis that attempt to reproduce critical characteristics of the client request stream. The majority of these tools use virtual clients that are connected to the tested server by a low round-trip time, high-speed local area network (LAN). Earlier tools, such as SPECWeb96 [37], emulated resource size, type, and popularity distributions based on logs of many popular web sites.

Hu et al. [16] analyzed the performance of an Apache 1.2 web server using SPECWeb96. They showed that the majority of the CPU time is spent in network processing. The authors proposed several optimization techniques, including using the *mmap* system call for file reading, URL caching, and logging optimization. Many of the proposed techniques were incorporated into future versions of Apache.

The importance of reproducing such characteristics as arrival time distribution and temporal locality of requests was shown in Barford et al. [9]. They developed a SURGE workload generation tool that reproduced the following real-life workload parameters: server resource size distribution, request size distribution, relative file popularity, embedded resource references, temporal locality of references and idle periods of individual users. They reported an increase of almost 3 times in average server CPU utilization for the load generated by SURGE compared to SPECweb96 with about the same 25 requests per second average request rate.

Banga et al. [5] describes the problem of workload emulation for a web server that is close to saturation. Most tools wait for a request to complete before issuing a new request. Therefore, as server request latency increases, the request arrival rate decreases, which occurs when a server is saturated. This effectively throttles the maximum achievable request rate for the workload generation tool. The authors developed their own Scalable Client (s-client) that eliminates the dependency on the server request latency.

A different approach to web server performance analysis is the study of live web servers under real-life workload. One of the earliest such studies analyzed the 1994 California Election web site [26]. This study identified the performance problem related to socket lookup and to the TIME_WAIT state handling. Fixes to these problems were incorporated in the operating systems [20][25].

Maltzahn et al. [23] studied the behavior of an enterprise-level web proxy that runs two different types of proxy servers: CERN and Squid. It was shown that a web proxy that is close to saturation spends most of its CPU time in the kernel managing network connections and passing data. Although the older CERN proxy has numerous inefficiencies in its design, it performs almost as well and consumes as many resources as the more sophisticated Squid under real-life workload. This is in sharp contradiction with Chankhunthod et al. [13], who report an order of magnitude better performance by the Harvest proxy, the predecessor of Squid, over the CERN proxy but under a simulated workload in LAN environment.

Seshan et al. [35] studied the TCP behavior of a live web server. The authors discovered that TCP recovery algorithms are not efficient for HTTP traffic in the presence of packet losses and reordering. They also found that multiple connections to the same client use network bandwidth unfairly. This study suggests sharing TCP information, such as congestion window control variables, among multiple connections to the same client to alleviate described problems.

Live web server studies identified many problems that could not be discovered using an emulated workload in a LAN environment. At the same time, these studies are hard to reproduce due to the uniqueness of the test environment. It is also hard to modify the test conditions to study the range of parameters of interest. This led to attempts to incorporate WAN conditions into the workload generator tools.

The Wide Area Web Management project [10] introduced WAN network conditions in the test environment by connecting SURGE clients and the test server by a wide area network. This approach captures the real-life network parameters, but it was hard to generate a high workload using it. Additionally, it partially suffers from the same problem as the live web server studies, which is the difficulty in changing the network parameters for different test scenarios.

The new version of SPECWeb96, SPECWeb99 [37], added dynamic content workload and attempted to emulate network delays by reading HTTP response in chunks. The latter method emulates the large number of concurrent connections opened to the server but fails to reproduce the network behavior of a real-life server due to a different mechanism for controlling the connection throughput.

The WASP project [29] emulates WAN conditions by introducing packet delays, reordering, and losses through a use of a kernel module at the clients. The authors discovered that the packet losses may decrease the server capacity by 50%. They also observed that the server had different performance properties under the WAN environment as compared to the LAN environment. This study focused on the behavior of the web server as it is visible to the clients and did not look into the internal behavior of the

web server at the kernel and network protocol level beyond studying effects of using different versions of TCP, such as SACK and New Reno.

The last approach is similar to the one used in this work for web server studies. It reproduces parameters that are necessary for a realistic study of web server performance and at the same time allows easy changes in test conditions. This approach lets us study a wide range of different workload parameters, which would be hard to do in real-life experiments.

III. WEB SERVER OPTIMIZATIONS

There have been many attempts to optimize HTTP request processing by changing the design of the HTTP servers and modifying operating system and network protocol processing. Most of these attempts evaluate the efficiency of the proposed optimizations using benchmarks in LAN environment. This section summarizes known optimizations in order to evaluate their efficiency under WAN workload later in the paper.

A. Concurrency Management

There are two widely used designs for HTTP servers: process-based and event-based. The former allocates a process (or a thread) to handle each incoming request. A process blocks in the operating system when some operation cannot be completed immediately, e.g., reading a file. This design is used by many popular servers, including the most popular, Apache [4]. To decrease the overhead of process scheduling, which can be high for a loaded server, a web server can be built using the event-based model where one or a limited number of processes handle socket events for all client requests. In this case, the operating system has to support non-blocking operations so that an operation that cannot be completed immediately would not block the process from processing events for other requests. For event dispatching, such servers usually use *select* or *poll* system calls. This design is becoming dominant these days for commercial web servers. An example of this design is the TUX HTTP server [38].

Pai et al. looked into the performance of different server architectures in the context of one implementation: the Flash web server [31]. This server uses a new architecture, which combines event-based processing for requests that hit the memory cache and process-based handling of the file system accesses. The study shows that event-based servers perform several times better than process-based servers for requests that hit memory cache. The Flash architecture performs as well as the event-based approach when the file working set fits in the memory cache and better than the other architectures when most of the requests are served from the file system.

To alleviate the overhead of event dispatching in web servers, such as *select* system call overhead, the implementation of this system call can be modified to avoid unnecessary scanning of the file descriptors that are known to be not ready at the time of the scan [8]. Another approach is to use a different event dispatching mechanism. Chandra et al. [12] suggested using POSIX.4 Real Time signals, which are more efficient than *select* or */dev/pool*. The latter study also claims that even an inefficient *select* has low overhead for a loaded web server because its cost is amortized over many requests. This claim is supported by our study.

The APFA platform [19] attempts to decrease the overhead of event dispatching even further by performing HTTP request processing in the socket event handler in software interrupt context. This platform also implements many of the optimizations described further in this section, including data copy/checksum avoidance and minimizing context switches by processing in the kernel.

B. Context Switches and Integration

To decrease the overhead of context switches, HTTP servers can be implemented as kernel daemons [24][38][39]. This design may also allow tighter integration of the server with the operating system and the network protocol stack. For example, a kernel HTTP server may directly access the file cache, which may not be possible from the user level. A different approach for tighter integration with the operating system is to provide specialized system calls that are designed with web servers in mind [6].

Another optimization that decreases the overhead of the context switches is reducing the number of hardware and software interrupts. Interrupt coalescing by a network adapter reduces the number of hardware interrupts increasing the minimum time between consecutive interrupts [2]. Some research suggests using device driver polling by periodically checking the network adapter for arrived or sent packets [14][28]. In this case, all network processing is performed in process context instead of interrupt context. This optimization is beneficial only for specialized environments that have high bulk transfer rate, such as a busy web server. For interactive applications, this optimization may introduce high latency.

The Lava caching server shows the effect of using a specialized operating system and building the server as integrated software [22]. The caching server, which can serve different types of objects besides web documents, implements a specialized network protocol as well as a general caching mechanism. It achieves throughput of 7000 requests per second for 10KB objects, almost an order of magnitude better than research web servers on the same hardware.

C. Data Copy and Checksumming

Several studies identified data copying and checksumming in the network protocol stack as the significant overhead for web servers [30][33]. BSD and Linux 2.4 kernels introduce a zero-copy framework, where data copying and checksumming during socket send operation are avoided by using scatter/gather and hardware checksumming capabilities of a network adapter. IO-Lite goes further by introducing a unified I/O buffering and caching system [30]. It allows applications, inter-process communication, the filesystem, the file cache, and the network subsystem to share a single physical copy of data. It also relies on the hardware capabilities of a network adapter for network zero-copy operations. Protection and security are maintained through a combination of access control and read-only sharing.

D. TCP Connection Management

Several studies claimed that TCP connection open and close operations for HTTP servers are computationally expensive [21][27][33]. With HTTP/1.1 persistent connections, it became possible to reduce the number of these operations by sending more than one request per connection. This decreases request processing overhead by avoiding expensive code paths in TCP module and by decreasing the number of packets processed.

E. HTTP processing Optimizations

Optimizing several operations on the HTTP processing path can reduce processing overhead. They include resource caching, HTTP header caching, resolved URL caching [31] and optimizations of the logging process [16].

F. Computation Offloading

Offloading computationally intensive operations to a network adapter with on-board processors can significantly improve web server performance. For example, the Alachritech [2] network adapter is capable of performing packet fragmentation and de-fragmentation, TCP acknowledgement processing, partial TCP connection management, and data copying directly to user-level buffers. This is in addition to performing more standard operations, such as checksum computation and interrupt coalescing. Adaptec TCP/IP offload adapter is capable of complete TCP/IP protocol offloading to the on-board processor [1].

It is possible to decrease the network protocol processing overhead by pre-processing client requests at a front-end booster appliance. This appliance changes the network characteristics of the requests, such as MTU size, and sends all requests over small number of persistent connections opened to the server. These techniques, along with several others, decrease the processing overhead up to an order of magnitude [32][33].

Another approach that removes the overhead of network protocol processing from a web server is migrating a communication link between the web server and the front-end appliance of a web site from the existing LAN technologies, such as Ethernet, to a System Area Network (SAN) architecture [15][18][36]. This architecture provides reliable communication in hardware and uses a computationally light-weight transport protocol, which is different from TCP. This approach requires the front-end

appliance to accept client TCP connections and forward data to and from the web server using the SAN transport protocol.

IV. EXPERIMENTAL METHODOLOGY

The experiments described in this paper use several tools that emulate client workload. The first one was written for micro-benchmarking. It requests a single object from the server with the constant request rate.

The second tool is used for generating realistic client workload. It was built from the SURGE workload generation tool [9]. SURGE emulates the statistical distribution of the following real-life workload parameters: (i) *server object size* distribution using lognormal model for the body of the distribution and Pareto model for the tail, (ii) *request size* distribution using Pareto model, (iii) *relative object popularity* using Zipf model, (iv) *embedded object references* using Pareto model, (v) *temporal locality of references* using lognormal model, and (vi) *idle periods* of individual users using Pareto model. It has been shown [9] that SURGE emulates real-life characteristics of client requests better than the standard tools, such as SPECWeb [37], particularly self-similarity of the server network traffic produced by the requests.

SURGE was modified for these experiments in two ways. First, the process-based design was changed to an event-based design in order to increase the maximum number of simulated clients. Second, the server throttling problem described by Banga et al. [5] was addressed. This problem shows up for an overloaded server when the higher processing latency of the requests delays the next request generated by the emulation tool, which waits for the completion of the previous request, and thus effectively decreases the request rate of the simulation. SURGE was modified not to wait for the current request completion to calculate the time for the issuing of the next request.

The third tool, which was written for these experiments, emulates network delays and bandwidth limits of the client connections. This tool is based on the Linux kernel and its TCP/IP stack. The receive path of the network protocol stack was modified to delay packets based on the emulated network parameters. Such packets are put on the queues that are distinct for each TCP socket. The regular Linux timers schedule the delivery or the transmission of the delayed packets with 10 ms granularity. Similar experiments [29] that used the Dummynet tool [34] indicate that 10ms granularity is precise enough to emulate network effects for the web server performance studies. Unlike these studies, which can only emulate network characteristics per network interface, our tool allows emulating parameters for each TCP connection. This is important for emulation of the bandwidth limit to clients, which is impossible to emulate with tools like Dummynet and NISTnet [11] because the bandwidth bottleneck is usually near the client and not at the network link to the server. The limitation of our tool is that it currently does not support the emulation of packet losses and reordering.

The fourth tool performs web server profiling. It measures time spent in different parts of the Linux kernel using CPU hardware counters. This tool accounts for context switches as well as for hardware and software interrupt processing. It uses breakpoints in the code to measure the time spent in a particular code path including all functions invoked in this path. This method allows measuring the time spent in all call sub-graphs of interest, not just time spent in a particular function regardless of the source of its invocation, as statistical sampling tools do [3]. The overhead of such profiling was measured by setting a dummy pair of breakpoints to start and stop the timer. These breakpoints were set in the actual code to measure such effects as processor cache misses between the successive breakpoint invocations. At a rate of about 20,000 invocations of this breakpoint pair per second, the total overhead of profiling was about 0.08% of the total time spend in processing.

Two HTTP servers were used for the experiments. The first one was Apache, which has the largest installation base [4]. This server is built using the process-based model and does not utilize any of the optimizations mentioned in the previous section. It is reported to be one of the worst performing among publicly and commercially available HTTP servers [19]. Nevertheless, due to its popularity it was included in our study.

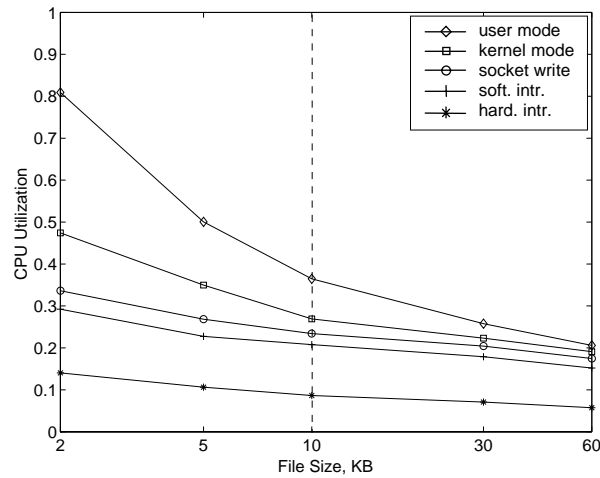


Figure 1. Cost breakdown vs. file size (Apache)

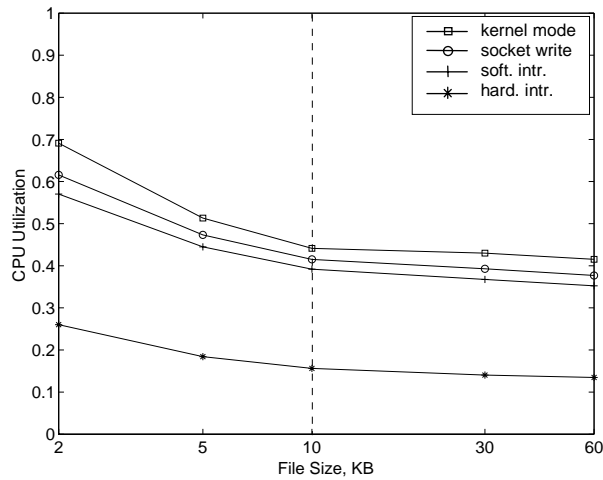


Figure 2. Cost breakdown vs. file size (TUX)

The second server is TUX, which is available for the Linux 2.4 kernel [38]. It is built as a high-performance, event-based kernel daemon. It deploys some of the optimizations described above, such as copy/checksum avoidance, kernel-level processing and object and name caching. It is reported to be one of the best performing among web servers [19].

All experiments study server performance for memory-cached objects by warming up the object or file cache with the objects before running the tests. The SURGE tool is configured with 146MB total size of the object set. The memory size of the test machines, which is 512MB, allows fitting the whole object set into the memory and making no file system accesses. These experiments cover the common case for commercial web sites, which install large main memory on front-end web servers to avoid file system accesses for the working set of objects.

Experiments use one server and two client machines of the same configuration. They are all Intel PIII 650MHz with 512MB of main memory running Linux 2.4 kernel. Each client was connected by one 3COM 3C590 100 Mbps network adapter to the server with the same type of the network adapter.

TABLE I
HTTP SERVER STATISTICS, 10KB FILE SIZE

Server	Apache	TUX
Server send rate, MB/s	3.0	6.0
# packets received / s	5738	11991
# packets sent / s	6156	11878
# interrupts / s	7482	13974
# concurrent connections	784	1451

V. RESULTS

This section presents the results of the web server performance study. First, it shows the distribution of CPU time in different parts of a web server as a function of the requested object size and the server load. Next, it breaks down the request processing cost in different parts of the network protocol stack and its dependency on the MTU size of the client connections. This is followed by a study of the effects of persistent connections on web server processing as well as the effects of network delay and connection bandwidth limit. Next, the section shows the effects of zero-copy and interrupt coalescing techniques. Finally, the measurements of the end-user request latency as a function of different network parameters are given.

A. Object Size

The first set of experiments studies a distribution of the CPU time spent in different parts of the kernel and HTTP server. The dependency of this distribution on the object size is measured using the micro-benchmarking tool mentioned in the previous section. This tool generates requests with a constant rate to a single object without using persistent connections. The experiments vary the size of the object from 2KB to 60KB. The connection bandwidth was fixed at 56 kbps per connection, network delay at 200 ms and the MTU size at 536 bytes, which are typical parameters for modem dial-up user.

Figure 1 and Figure 2 plot the fraction of CPU time spent in different parts of the kernel and HTTP server vs. file size for experiments with Apache and TUX. The data send rate is 3 MB/s for Apache and 6 MB/s for TUX, in both cases to keep the CPU utilization close to 50% for 10 KB file size. This makes the request rate for TUX twice as high as for Apache on these graphs. Because the data send rate in the experiments with the different file sizes is only approximately the same, the CPU utilization on the graphs is normalized to the abovementioned constant data send rates.

These graphs, as well as graphs in the rest of this section, show a cumulative stack of the components, where each curve is the sum of the plotted component and the all components plotted below it. There are five components plotted: the fraction of time spent in hardware interrupts, in software interrupts, in process context in kernel mode, in user mode and, as a separate item, the time spent in the socket *write* system call in the process context. The last component is a part of the kernel-level processing in process context, but is graphed separately.

Because Apache runs in the user level, user time for the Apache experiments covers such operations as HTTP header processing and name resolution. The time spent in the kernel in process context includes network processing (mostly for data sending) as well as the concurrency management overhead. In the TUX experiments, all time is spent in the kernel mode because TUX is implemented as the kernel daemon.

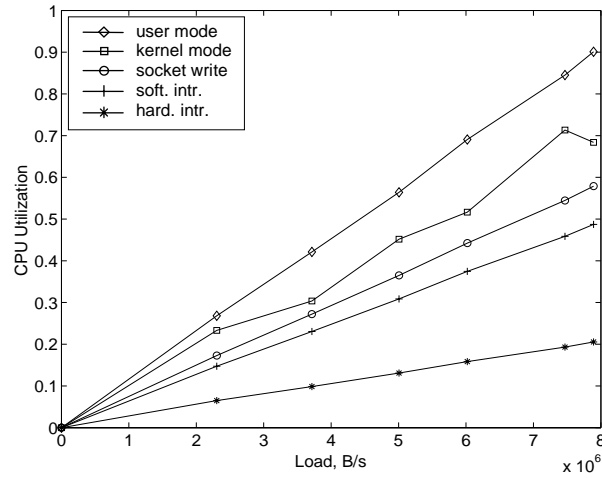


Figure 3. Load scaling (Apache)

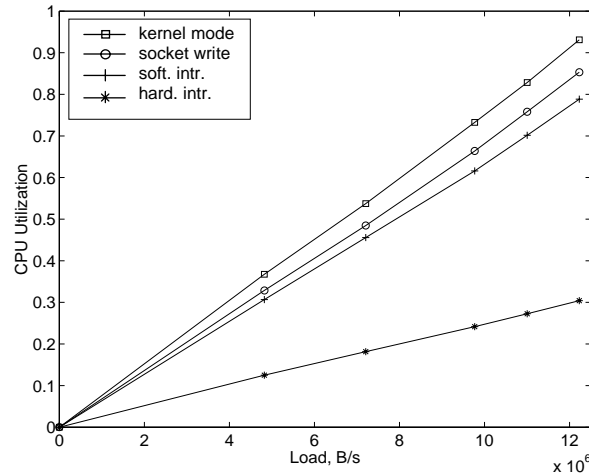


Figure 4. Load scaling (TUX)

We can see that in spite of the fact that most of the server data are sent rather than received, most of the time for object sizes larger than 10 KB is spent in hardware and software interrupt handling, which is triggered by packet reception. From Table I, which shows experimental statistics for tests with 10KB object size, we can see that the number of sent and received packets is approximately the same. Such a high number of received packets, which are mostly acknowledgments, shows that the TCP connections are in the slow-start phase during most of their lifetime, when acknowledgments from the peer sockets are not delayed [35].

Different design choices for HTTP servers affect the relative time spent in the HTTP server and all system calls. Figure 1 and Figure 2 show that for a 10KB object size, this time is small for TUX (upper two components on Figure 2) but is almost half of the overall cost for Apache (upper three components on Figure 1). As described previously, Apache uses a process-based model and runs at the user level, while TUX uses an event-based model and implemented as a kernel daemon. Unfortunately, due to significant differences in implementation of servers, particularly in HTTP-related processing path, the experiments do not let us precisely attribute the difference in the abovementioned costs to the difference in the concurrency model or to a choice of user or kernel level execution mode.

At the same time, the difference in the cost of the kernel mode processing in process context for Apache and TUX (excluding socket write cost) may indicate the operating system overhead of the process

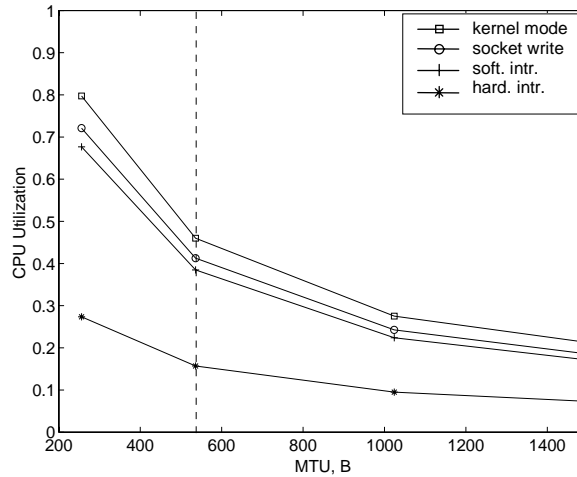


Figure 5. Cost breakdown vs. MTU (TUX)

management and the cost of the context switches between the user and the kernel mode. This is because the rest of the kernel mode processing in process context is spend in different system calls, which are similar for Apache and TUX. Cost of the kernel mode processing in process context is almost three times higher for Apache than for TUX. Figure 2 also shows that optimizing HTTP request processing and improving concurrency management, i.e. optimizing *select* call [8][12] or doing processing at the socket event handler [19], would not significantly improve performance under the measured workload due to already low relative cost of such overhead.

B. Load Scaling

The next set of experiments studies how the server utilization scales with load increase. Figure 3 and Figure 4 show the server utilization for Apache and TUX as a function of the server load. The SURGE toolkit is used to generate requests with the distribution of object sizes according to the SURGE model. The average object size is 10KB and the network parameters are the same as for the previous set of experiments. The graphs show a different CPU time distribution under the SURGE workload load compared to the previous experiments that generated requests to a single 10 KB object.

In these experiments, Apache achieved 8MB/s data send rate while TUX achieved about 13MB/s rate, which correspond to about 800 and 1300 requests per second respectively for the 10KB average object size. CPU utilization is brought to almost 95% before both servers stopped responding to new requests. Graphs show that the CPU utilization scales linearly with the load defined by the send data rate. The exception to this is the user and kernel level components of the cost for Apache. This seems to be the artifact of the process-based design of Apache. It maintains a pool of working threads and increases the pool size in increments when load increases above some threshold.

C. MTU size

Figure 5 shows the dependency of the request processing cost on the MTU size of the incoming client connections for TUX server. For these experiments, SURGE generates requests with an average object size of 10 KB. Network parameters are set at the base values of 200 ms for the network delay and 56 kbps for the connection bandwidth limit. The graph shows that an increase in MTU size from 256B to 1500B decreases the overhead by almost factor of four. Most of the overhead is proportional to the number of processed packets, which is inversely proportional to the MTU size. This makes hardware and software interrupt components decrease proportionally to each other with MTU increase.

To look closer at what is happening in the network protocol stack, the time spent by TUX in software interrupts was further subdivided into the time spent in the code path for the TCP ESTABLISHED state, TCP FIN_WAIT_1 state, connection open/close and the rest of the code. Figure 6 shows the dependency

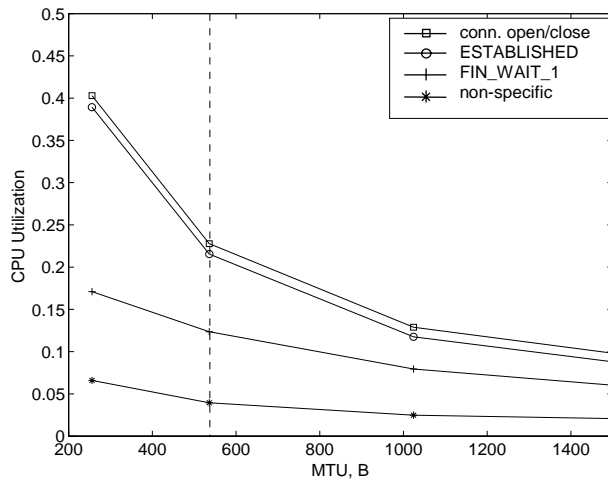


Figure 6. Receive processing vs. MTU (TUX)

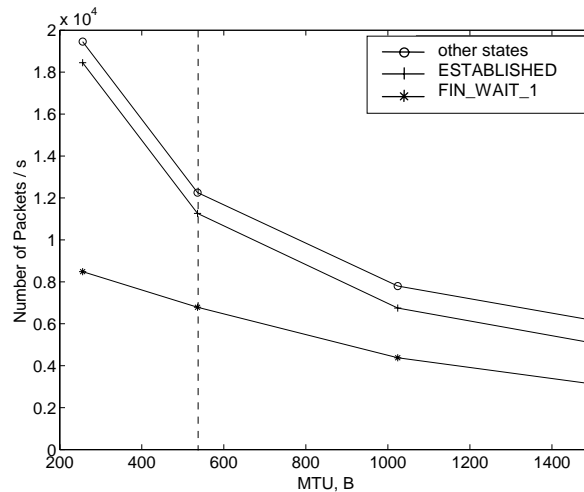


Figure 7. Number of packets vs. MTU (TUX)

of these components on the MTU size. Figure 7 shows the number of received packets per second that are processed in those states vs. the MTU size. As we can see, more than half of the received packets are processed in the FIN_WAIT_1 state, except at the smallest MTU.

This processing can be explained considering that the socket write buffer in the experiments was 16KB, which is higher than the average size of the objects. When the HTTP server writes an object into the socket, it is queued at the buffer rather than sent immediately due to the TCP slow start. Because the object fits in the buffer, the HTTP server closes the socket and the socket switches into the FIN_WAIT_1 state. The majority of the received packets are acknowledgements and they are processed in FIN_WAIT_1 state, controlling the sending of queued data.

D. Persistent Connections

To evaluate the effect of HTTP/1.1 persistent connections and the overhead of the TCP connection management on the request processing cost, we measure the TUX server performance with persistent connections enabled. SURGE generates the same distribution of the requests as in the previous experiments with multiple requests delivered on the same connection. The experiments vary the number of requests per connection from 1 to 16 by sending the “Connection: close” header on the last request, causing the server to close the connection after delivering the last response. The network parameters were kept at the same base values as in previous experiments and the MTU size was 536 B.

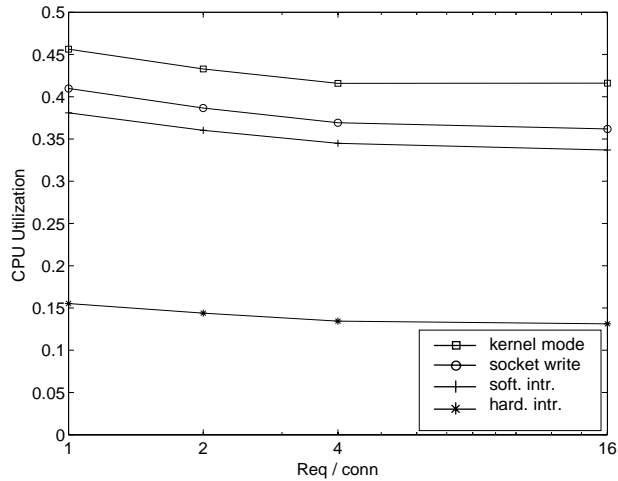


Figure 8. Cost breakdown for persistent connections (TUX)

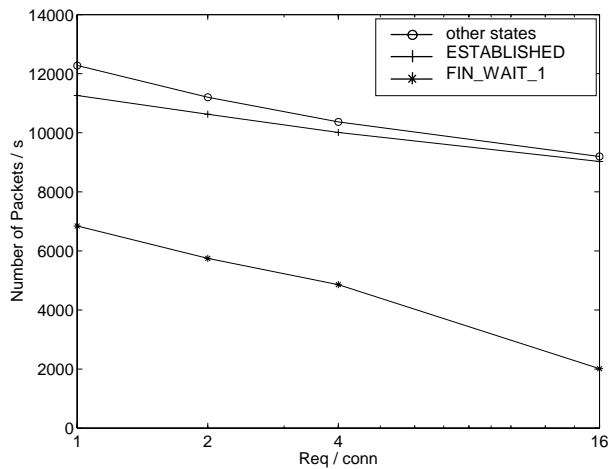


Figure 9. Number of packets for persistent connections (TUX)

Figure 8 shows the cost breakdown for TUX as a function of the number of requests per one connection. The effect of introducing the persistent connections is not high. The overhead is decreased by about 10% compared to opening a separate connection for every request. Most of the decrease is due to the processing in hardware and software interrupts, while kernel mode process context and socket write components stay about the same.

Figure 9 shows the number of packets processed in different TCP states vs. number of requests per connection for the same set of experiments. The graph shows that the number of packets processed decreases by about 25% for 16 requests per connections compared to the base HTTP/1.0 case. Part of this decrease is due to the almost complete elimination of the connection open and close packets. A decrease in the number of packets processed in other TCP states can be explained by a smaller fraction of connection lifetime spent in the TCP slow-start phase. This enables delayed acknowledgments from the client, which decreases the number of acknowledgements processed by the server. The graph also shows that the number of packets processed in the FIN_WAIT_1 state is significantly lower than in the ESTABLISHED state for 16 requests per connection.

Figure 10 subdivides the time spent by TUX in software interrupts into the time spent in the code path for the TCP ESTABLISHED state, TCP FIN_WAIT_1 state, connection open/close and the rest of the code. As expected, persistent connections bring the cost of connection open and close to almost zero.

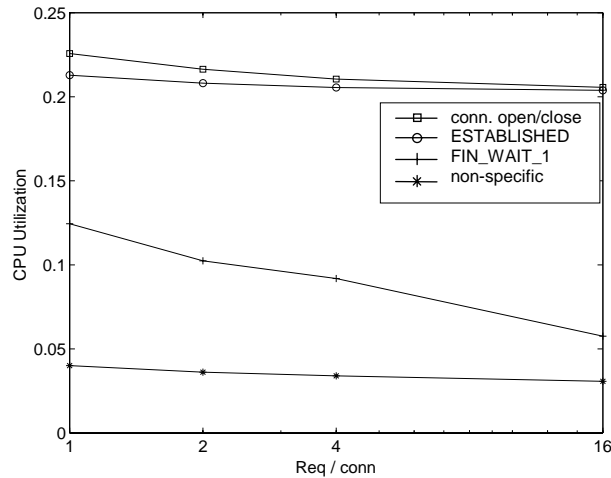


Figure 10. Receive processing for persistent connections (TUX)

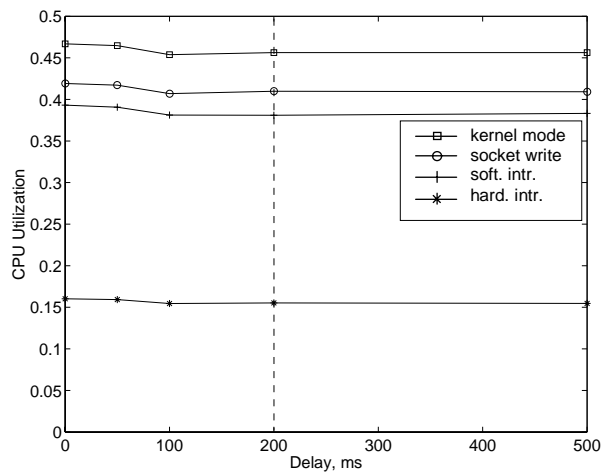


Figure 11. Cost breakdown vs. delay (TUX)

Although the number of packets processed in the ESTABLISHED and FIN_WAIT_1 states combined decrease by about 20% for 16 requests per connection, the overhead of processing in these two states combined stays about the same when compared to the base case. Because the majority of packets are processed in the ESTABLISHED state for 16 requests per connection, the cost of processing in the ESTABLISHED state is higher than that in the FIN_WAIT_1 state.

The low effect of persistent connections on processing overhead contradicts previous studies [21]. Saving CPU time by opening and closing fewer TCP connections is also listed as one of the advantages for HTTP/1.1 persistent connections in specification of the protocol [17]. The majority of these studies measure this effect in a LAN environment with a large MTU size, where the number of connection open and close packets is comparable to the number of data and acknowledgement packets. The workload used for the experiments described above makes the number of connection open and close packets and associated overhead low.

E. Network Delay and Bandwidth Limit

The next set of experiments measure the dependency of the server processing cost on the network parameters. Figure 11 shows the dependency of the TUX processing cost on the network delay and Figure 12 shows dependency of the cost on the connection bandwidth limit. In each test, one of these parameters

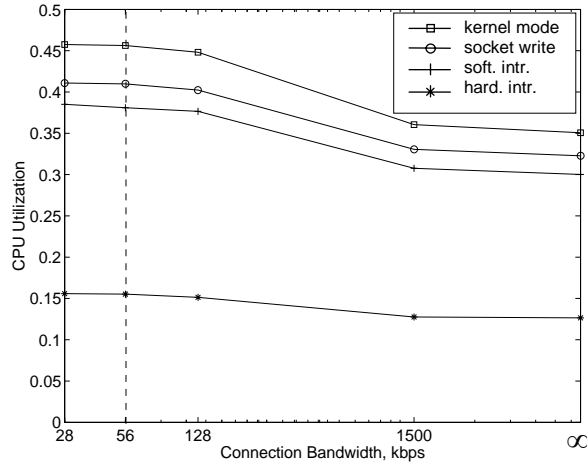


Figure 12. Cost breakdown vs. bandwidth (TUX)

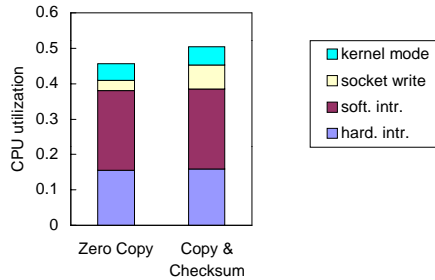


Figure 13. Effects of copying and checksumming (TUX)

is varied while the other is kept constant with the delay set at 200 ms, the bandwidth limit at 56 kbps. The MTU size was set at 536 bytes for both tests. SURGE is used to generate the workload with a 10KB average size of the objects. These graphs show that effect of network delay on server performance is insignificant. Limiting the connection bandwidth increases the overhead by approximately 20%.

F. Data Copying and Checksumming

To evaluate the effect of data copying and checksumming during the socket send operation, the performance of TUX is measured in two configurations: the normal configuration used in this study, which avoids these operations by using scatter/gather network buffers and checksumming by the network adapter, and a configuration where these capabilities are disabled and data are copied and checksummed in a regular way. Figure 13 shows the cost breakdown for these two configurations under the SURGE workload with the base network parameters used throughout these experiments. The graph indicates that the effect of copy/checksum avoidance is only about 10% for the workload used in the experiments. This is contrary to a popular belief that such avoidance significantly increases web server performance [19][30].

G. Interrupt Coalescing

To study the effect of interrupt coalescing, we ran experiments with Apache and TUX with coalescing enabled. SURGE generates a workload with 10KB average object size and the base network parameters used throughout these experiments. Figure 14 shows the cost breakdown for Apache and TUX with and without interrupt coalescing. The graphs are normalized to keep the CPU utilization at 50% for the base

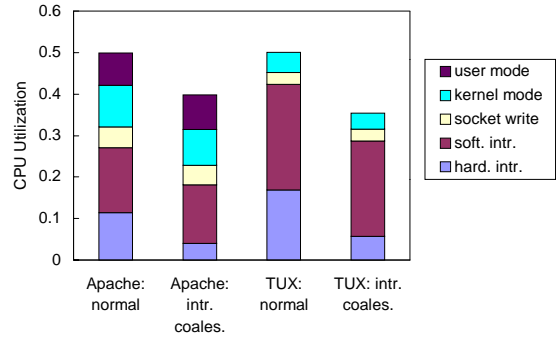


Figure 14. Interrupt coalescing effects

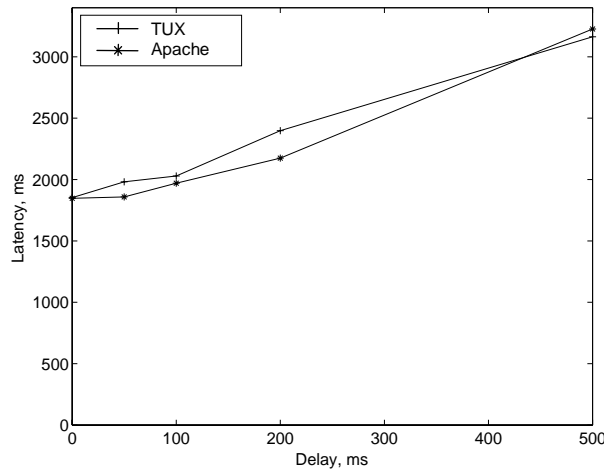


Figure 15. Latency vs. delay

cases for both servers, which makes the workload for TUX higher than for Apache. For both Apache and TUX, the interrupt rate is decreased to about 520 interrupts per second from the values listed in the Table I. Interrupt coalescing decreases the hardware interrupt cost by nearly a factor of three. It has a higher overall effect for TUX and decreases the total processing overhead by about 25%.

H. Request Latency

Figure 15, Figure 16 and Figure 17 show how end-user request latency changes with variations in the network parameters. The latency was measured as the time from the client calling *connect* until the last byte of the response is received. Such latency includes the client network protocol latency. These are the same experiments shown in Figure 11, Figure 12, and Figure 5 where each of the three emulated network parameters was varied. In addition to TUX, the same measurements were performed for Apache. The graphs show that request latency is about the same for both servers.

MTU size has little effect on the request latency. Changing the network delay from 0 to 500 ms increases the request latency by about 80%. On the other hand, changes in the connection bandwidth limit from 28 kbps to infinity change the latency by almost factor of six. These data show that delay in the network limits the throughput of the TCP connections only moderately. The latency is mostly affected by the connection open/close and a few initial packets of response that require acknowledgements from the other side before new data can be sent. These packets introduce round-trip time delay for each packet. The rest of the data are pipelined so delay does not have much effect.

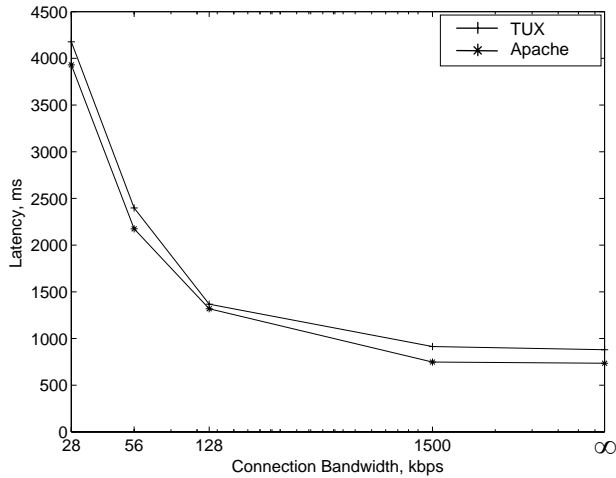


Figure 16. Latency vs. bandwidth

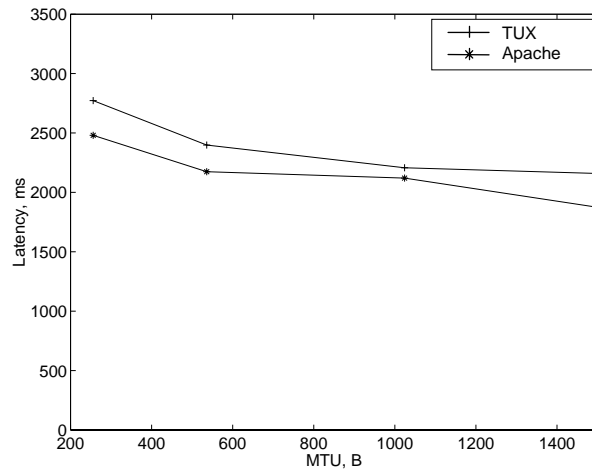


Figure 17. Latency vs. MTU

On the other hand, connection bandwidth limits have a direct effect on the TCP connection throughput. For a 10KB object and 28 kbps bandwidth, the time to transfer the data is about 3 seconds, which is close to the 4 seconds latency shown on the graph.

VI. CONCLUSION

This paper studied the performance of the two HTTP servers, Apache and TUX, under workload in simulated WAN environment. The results of the study indicate that for both servers, network processing in hardware and software interrupts is a significant part of the overall cost. This confirms previously published results that were obtained using less realistic workload simulations [16]. Our measurements discover that the processing overhead is highly dependent on the MTU size and much less dependent on the network delay and connection bandwidth limit or using HTTP/1.1 persistent connections. At the same time, the request latency mostly depends on the connection bandwidth limit and, to some extent, on the network delay. Experiments showed that a significant part of the network processing is accomplished in TCP FIN_WAIT_1 state if persistent connections are not used, and in ESTABLISHED state otherwise.

We summarize the effects of known web server optimizations listed in Section 3 that have been evaluated in this paper. Locating a server in the kernel and migrating from a process-based design to an event-based design may lead to a significant performance improvement. At the same time, further optimizations related to concurrency management, such as optimizing *select* call [8][12] or doing processing at the socket event handler [19], would not introduce significant performance differences due to already low relative cost of such overhead. HTTP processing optimizations implemented in TUX made the cost of HTTP processing small comparatively to other component costs. Both the HTTP/1.1 persistent connections and copy/checksum avoidance do not have significant effect on the server processing overhead. Interrupt coalescing provides only moderate improvements. These conclusions are reached for the workload used for the experiments in this study, particularly MTU size of the client connection in these experiments is small. At the same time, some of the mentioned optimizations may show significant effects in a LAN environment.

Results of this paper show that failure to reproduce network characteristics of a WAN environment in web server performance studies can lead to significant differences in obtained results and server behavior in real world. Such characteristics have to be incorporated into the standard benchmarks, most of which do not emulate them at this time.

REFERENCES

- [1] Adaptec Inc. TCP/IP Offload Adapter ANA-7711. <http://www.adaptec.com>.
- [2] Alacritech Inc. <http://www.alacritech.com>.
- [3] J. M. Anderson, L. M. Berc, J. Dean, S. Ghemawat, M. R. Henzinger, S. A. Leung, R. L. Sites, M. T. Vandervoorde, C. A. Waldspurger, and W. E. Weihl. Continuous Profiling: Where Have All the Cycles Gone? In *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP)*, Saint-Malo, France, October 1997.
- [4] Apache Web Server. <http://www.apache.org>.
- [5] G. Banga and P. Druschel. Measuring the Capacity of a Web Server. In *Proceedings of the 1997 USENIX Symposium on Internet Technologies and Systems*, Dec. 1997.
- [6] G. Banga, P. Druschel, and J. C. Mogul. Better Operating System Features for Faster Network Servers. In *Proceedings of the Workshop on Internet Server Performance (held in conjunction with ACM SIGMETRICS '98)*. June 1998.
- [7] G. Banga and J. C. Mogul. Scalable Kernel Performance for Internet Servers under Realistic Loads. In *Proceedings of the USENIX Annual Technical Conference*, New Orleans, LA, June 1998.
- [8] G. Banga, J. C. Mogul, and P. Druschel. A Scalable and Explicit Event Delivery Mechanism for UNIX. In *Proceedings of 1999 USENIX Annual Technical Conference*, 1999.
- [9] P. Barford and M. E. Crovella. Generating Representative Web Workloads for Network and Server Performance Evaluation. In *Proceedings of Performance '98/ACM SIGMETRICS '98*, Madison, WI, 1998.
- [10] P. Barford and M. Crovella. Critical Path Analysis of TCP Transactions. In *Proceedings of ACM SIGCOMM Symposium on Communications Architectures and Protocols*, Aug. 2000.
- [11] M. Carson. NIST Net. <http://www.antd.nist.gov/nistnet>.
- [12] A. Chandra and D. Mosberger. Scalability for Linux Event-Dispatching Mechanisms. In *Proceedings of 2001 USENIX Annual Technical Conference*, June 2001.
- [13] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K.J. Worrell. A Hierarchical Internet Object Cache. In *Proceedings of the 1996 USENIX Technical Conference*, San Diego, CA, Jan. 1996.
- [14] B. Chen and R. Morris. Flexible Control of Parallelism in a Multiprocessor PC Router. In *Proceedings of 2001 USENIX Annual Technical Conference*, June 2001.
- [15] Giganet, Inc. Giganet cLAN Product Family. <http://www.giganet.com/products>.
- [16] Y. Hu, A. Nanda, Q. Yang. Measurement, Analysis and Performance Improvements of Apache Web Server. Technical Report 1097-0001, University of Rhode Island, RI, Oct. 1997.
- [17] Hypertext Transfer Protocol 1.1 Specification. RFC2068. <http://www.w3.org>.
- [18] Infiniband Architecture Specification. Vol. 1&2. Rel. 1.0. http://www.infinibandta.org/download_spec10.html.
- [19] P. Joubert, R. B. King, R. Neves, M. Russinovich, J. M. Tracey. High-Performance Memory-Based Web Servers: Kernel and User-Space Performance. In *Proceedings of 2001 USENIX Annual Technical Conference*, June 2001.

- [20] P. E. McKenney and K. F. Dove. Efficient Demultiplexing of Incoming TCP Packets. In *Proceedings of the SIGCOMM '92 Conference*, Aug. 1993.
- [21] E. Levy-Abegnoli, A. Iyengar, J. Song, and D. Dias. Design and Performance of a Web Server Accelerator. In *Proceedings of IEEE INFOCOM'99 Conference*, New York, NY, March 1999.
- [22] J. Liedtke, V. V. Panteleenko, T. Jaeger, and N. Islam. High-Performance Caching With the Lava Hit-Server. In *Proceedings of the 1998 USENIX Annual Technical Conference*, New Orleans, LA, June 1998.
- [23] C. Maltzahn, K. J. Richardson, and D. Grunwald. Performance Issues of Enterprise Level Web Proxies. In *Proceedings of the ACM SIGMETRICS '97 Conference*, Seattle, WA, June 1997.
- [24] Microsoft Corporation. Installation and Performance Tuning of Microsoft Scalable Web Cache (SWC 2.0). <http://www.microsoft.com/technet/iis/swc2.asp>.
- [25] J. C. Mogul. Operating System Support for Busy Internet Server. Technical Report Technical Note TN-49, Digital Western Research Laboratory, May 1995.
- [26] J. C. Mogul. Network Behavior of a Busy Web Server and its Clients. *Technical Report WRL 95/5*, DEC Western Research Laboratory, Palo Alto, CA, Oct. 1995.
- [27] J. C. Mogul. The Case for Persistent-connection HTTP. In *Proceedings of ACM SIGCOMM'95 Conference*, Oct. 1995.
- [28] J. C. Mogul and K. K. Ramakrishnan. Eliminating Receive Livelock in an Interrupt-driven Kernel. *ACM Transactions on Computer Systems*, 15(3), Aug. 1997.
- [29] E. M. Nahum, M. C. Rosu, S. Seshan, and J. Almeida. The Effects of Wide-Area Conditions on WWW Server Performance. In *Proceeding of ACM SIGMETRICS'00 Conference*. 2000.
- [30] V. S. Pai, P. Druschel, and W. Zwaenepoel. IO-Lite: A Unified I/O Buffering and Caching System. In *Proceedings of the 3rd Symposium on Operating System Design and Implementation*, New Orleans, LA, Feb. 1999.
- [31] V. S. Pai, P. Druschel, and W. Zwaenepoel. Flash: An Efficient and Portable Web Sever. In *Proceedings of the 1999 USENIX Annual Technical Conference*, Monterey, CA, June 1999.
- [32] V. V. Panteleenko. Instantaneous Offloading of Web Server Load. Ph.D. Dissertation, Department of Computer Science and Engineering, University of Notre Dame, 2002.
- [33] V. V. Panteleenko and V. W. Freeh. Instantaneous Offloading of Transient Web Server Load. In *Proceedings of the Sixth International Workshop on Web Caching and Content Distribution*, Boston, 2001.
- [34] L. Rizzo. Dumynet: A Simple Approach to the Evaluation of Network Protocols. In *Computer Communication Revue*, 27(2), Feb. 1997.
- [35] S. Seshan, H. Balakrishnan, V. N. Padmanabhan, M. Stemm, and R. H. Katz. TCP Behavior of a Busy Internet Server: Analysis and Improvements. In *Proceedings of IEEE Conference on Computer Communications (INFOCOM) '98*, San Francisco, CA, March 1998.
- [36] H. V. Shah, D. B. Minturn, A. Foong, G. L. McAlpine, R. S. Madukkarumukumana, and G. J. Regnier. CSP: A Novel System Architecture for Scalable Internet and Communication Services. In *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems (USITS'01)*, San Francisco, CA, March 2001.
- [37] The Standard Performance Evaluation Corporation. SPECWeb96/SPECWeb99. <http://www.spec.org/osg/>.
- [38] TUX Web Server 2.0. <http://www.redhat.com/products/software/tux>.
- [39] A. van de Ven. kHTTPd Linux HTTP accelerator. <http://www.fenrus.demon.nl>.