

Web Server Performance in a WAN Environment

Vsevolod V. Panteleenko and Vincent W. Freeh

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

Department of Computer Science
North Carolina State University
Raleigh, NC 27695
vin@cs.ncsu.edu

Abstract—This paper 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. We discovered that WAN network characteristics may significantly change the behavior of the web server compared to LAN-based simulations and may make many of optimizations of server design irrelevant in this environment. Particularly, we found that the small MTU size of the dial-up or wireless user connections can increase the processing overhead several times.

I. INTRODUCTION

Large public web sites are built in multiple tiers and serve significant amounts of dynamic content. Nevertheless, the performance of the first-tier web servers, which handle static content and serve as a front-end for dynamic content, is critical for overall site performance. Web server performance has been extensively studied in the literature [4][5][13][26]. Many of these studies use simulated workload in a local area network (LAN) environment. However, web servers deployed in the real world often show significantly different behavior from these studies [20][23][33]. 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) 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 and increasing numbers of wireless users, small packet (MTU) size of connections, and packet losses. Therefore, even though web server performance has been extensively studied, there is a need for more study.

In a recent paper, Nahum et. al. [26] considered the effect of more realistic network characteristics and workloads on web servers. In contrast to that work, which mainly looks at the characteristics visible to the clients, this work examines the internal behavior of a web server at the network protocol stack and the device driver levels under emulated WAN conditions. It also evaluates known techniques for server performance improvement in such an environment. Our experiments emulate a wide range of real-life workload characteristics, such as network delay, limited bandwidth of connections and small MTU sizes. We use two web servers for the experiments:

Apache [3] and TUX [36], both running on Linux operating system.

In our experiments we discovered the following.

- Similar to experiments in LAN environment [13], 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 do not have significant effect on the server performance in WAN environment.
- End-user request latency mostly depends on connection bandwidth limits 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 [16][27], (ii) optimizations of request concurrency management by optimizing *select* system call [6][10] or performing computations at the socket event handler [16], and (iii) connection open/close avoidance using HTTP/1.1 persistent connections [18][24]. This is contrary to claims made previously based on measurements in LAN environment.

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. Most of these tools use virtual clients that are connected to the tested server by a low round-trip time, high-speed LAN [13].

The importance of reproducing such characteristics as arrival time distribution and temporal locality of requests was shown in Barford et al. [7]. They developed SURGE, a workload generation tool that reproduces many real-life workload parameters, such as relative file popularity and idle periods of individual users. Authors report a nearly three-fold

increase in average server CPU utilization for the load generated by SURGE compared to SPECWeb96 [35].

One of the earliest studies of live web servers under real-life workload analyzed the 1994 California Election web site [23]. This study identified the performance problem related to socket lookup and to the TIME_WAIT state handling. Fixes to these problems were incorporated in operating systems [17][22].

A study of enterprise-level web proxy servers (CERN and Squid) shows that a web proxy that is close to saturation spends most of its CPU time in the kernel managing network connections and passing data [20]. Another study of a live web server discovered that TCP recovery algorithms are not efficient for HTTP traffic in the presence of packet losses and reordering [33].

The Wide Area Web Management project [8] introduced WAN network conditions in the test environment by connecting SURGE clients and the test server by a WAN. This approach captures the real-life network parameters, but it was hard to generate a high workload using it.

The WASP project [26] uses a kernel module at the clients that emulates WAN conditions by introducing packet delays, reordering, and losses. The authors discovered that packet losses might decrease the server capacity by 50%. They also note that the server had different performance properties under the WAN environment as compared to the LAN environment.

III. WEB SERVER OPTIMIZATIONS

This section details optimizations of HTTP request processing by changing the design of the HTTP servers and modifying operating system and network protocol processing. An event-based model, where one or a limited number of processes handle socket events for all client requests, can decrease the overhead of process scheduling. For event dispatching, such servers usually use *select* or *poll* system calls. To improve the performance, the implementation of these system calls can be modified to avoid unnecessary scanning of the file descriptors that are known to be not ready at the time of the scan [6]. Another approach is to use a different event dispatching mechanism. Chandra et al. [10] suggested using POSIX.4 Real Time signals.

Other approaches to improving the concurrency management are implementing an HTTP server as kernel daemons [21][36][37], performing HTTP request processing in the socket event handler in software interrupt context [16], building the server integrated with the operating system [19], using interrupt coalescing in the network device driver [2], and using device driver polling by periodically checking the network adapter for arrived or sent packets [11][25].

TCP connection open and close operations for HTTP servers can be computationally expensive [18][24][30]. With HTTP/1.1 persistent connections, it became possible to reduce the number of these operations by sending more than one request per connection. Optimizing several operations on the HTTP processing path can also reduce processing overhead. They include resource caching, HTTP header caching, resolved URL caching [28], and optimizations of the logging process [13]. In addition, data copying and checksumming in the

network protocol stack introduce a significant overhead [27][30]. Several network adapters with on-board processors [1][2] can offload these and other computationally intensive TCP/IP operations.

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 a 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 [29][30]. 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 [12][15][34].

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 [7]. SURGE emulates the statistical distribution of the server object sizes, request sizes, relative object popularity, embedded object references, temporal locality of references, and idle periods of individual users.

The third tool modifies the Linux kernel and TCP/IP stack to introduce network delays and bandwidth limits of the client connections. 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. This tool allows emulating parameters for each TCP connection, unlike Dummynet [32] and NISTnet [9] tools, which can only emulate network characteristics per network interface. This is important for emulation of the bandwidth limit to clients.

Two HTTP servers were used for the experiments. The first one was Apache, which has the largest installation base [3]. This server is built using the process-based model and does not utilize any of the optimizations mentioned in the previous section. The second server is TUX, which is available for the Linux 2.4 kernel [36]. It is a high-performance, event-based kernel daemon. It deploys many optimizations, such as copy/checksum avoidance, kernel-level processing, and object and name caching.

We profiled the kernel using CPU hardware counters. 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, while the memory size of the test machines is 512MB. These experiments cover the common case for commercial web sites, which install large

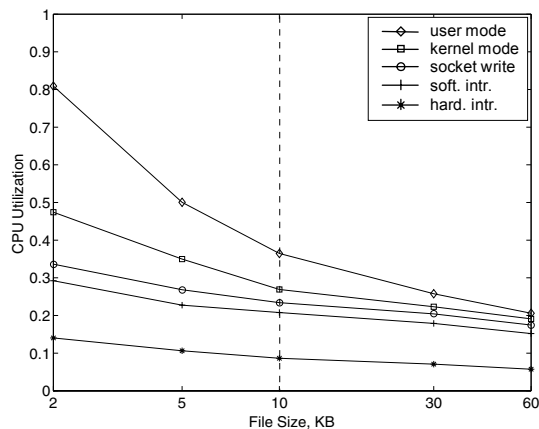


Figure 1. Cost breakdown vs. file size (Apache)

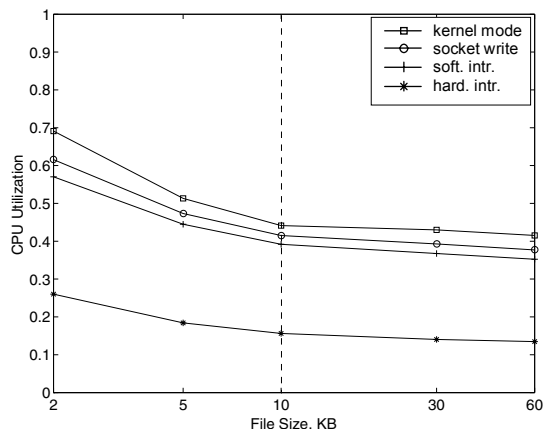


Figure 2. Cost breakdown vs. file size (TUX)

TABLE I
HTTP SERVER STATISTICS, 10KB FILE SIZE

Server	Apache	TUX
Server send rate, MB/s	3.0	6.0
No. of packets received / s	5738	11,991
No. of packets sent / s	6156	11,878
No. of interrupts / s	7482	13,974
No. of concurrent connections	784	1451

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.

V. RESULTS

This section presents the results of the web server performance study. In our experiments, we used a fixed base set of emulated network parameters when varying one of the parameters in a particular experiment. These were chosen as follows: MTU size is set to 536 bytes, connection bandwidth to 56 kbps, and the network delay to 200 ms, which are typical parameters for modem dial-up user. The data send rate is kept at 3 MB/s for Apache and 6 MB/s for TUX, in both cases to keep the CPU utilization close to 50%. Varying the level of workload showed that the server utilization scales linearly with the workload up to 95% of the utilization. Under the SURGE workload, Apache achieved 8MB/s maximum 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.

A. File 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. Other network parameters were set at the base levels.

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. 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.

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 [33].

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). The difference in the cost of the

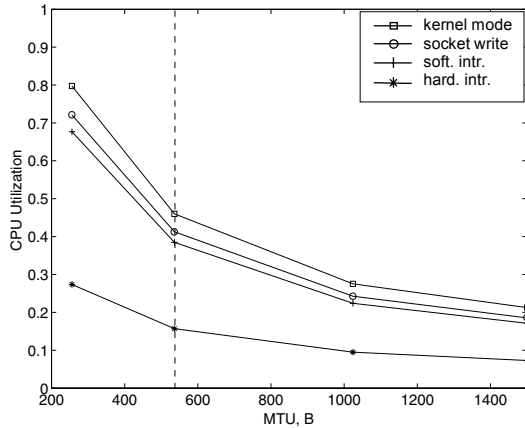


Figure 3. Cost breakdown vs. MTU (TUX)

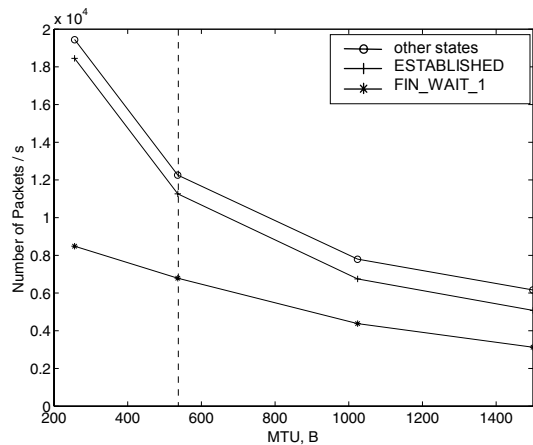


Figure 4. Number of packets vs. MTU (TUX)

kernel mode processing in process context for Apache and TUX (excluding socket write cost) indicate the operating system overhead of the process 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 [6][10] or doing processing at the socket event handler [16], would not significantly improve TUX performance under the measured workload due to already low relative cost of such overhead.

B. MTU size

Figure 3 shows the dependency of the request processing cost on the MTU size of the incoming client connections for TUX server¹. For these experiments, SURGE generates requests with an average object size of 10 KB and the network parameters (except MTU) are set at the base values. 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.

Figure 4 shows the number of received packets per second that are processed in different TCP 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.

C. 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 base values.

Figure 5 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 6 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 low effect of persistent connections on processing overhead contradicts previous studies [18]. 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

¹ Due to the space limitations, most of the graphs in the rest of the paper show only TUX experiments. Experiments with Apache showed similar behavior.

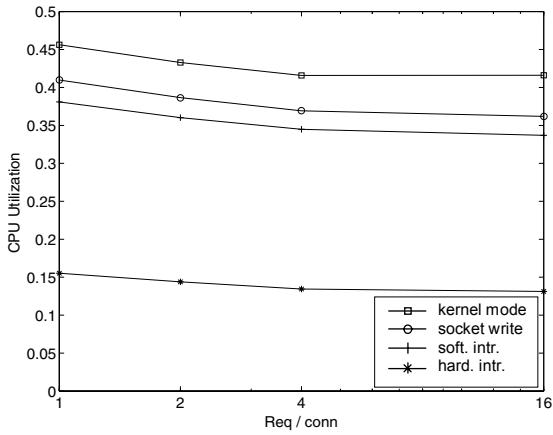


Figure 5. Cost breakdown for persistent connections (TUX)

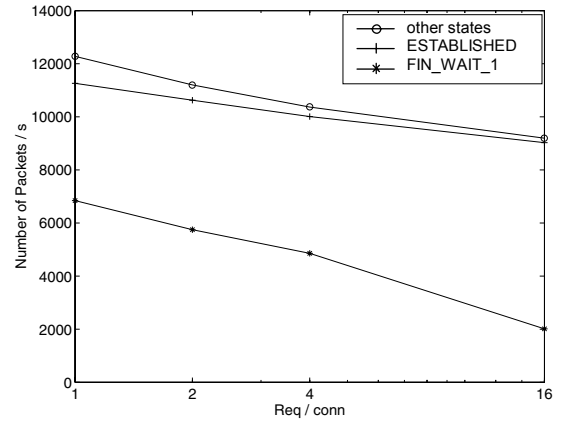


Figure 6. Number of packets for persistent connections (TUX)

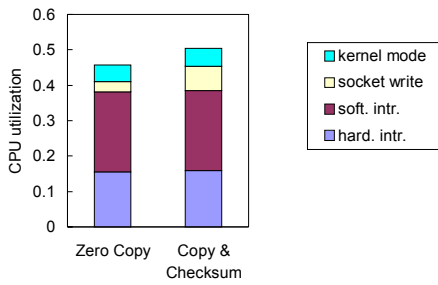


Figure 7. Effects of copying and checksumming (TUX)

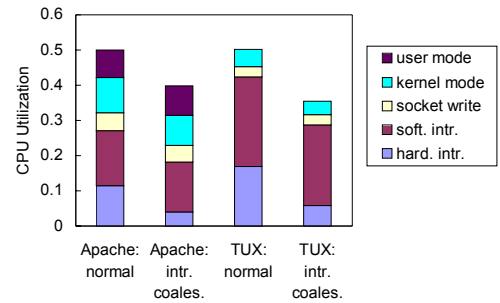


Figure 8. Interrupt coalescing effects

specification of the protocol [14]. 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 reduces the relative number of connection open and close packets and the associated overhead.

D. Network Delay and Bandwidth Limit

We measured the dependency of the server processing cost on the network parameters. Due to the space constraints, we only summarize the results (see [31] for complete results).

Both network delay and bandwidth limit have little effect on the performance of the servers. The network delay does not change the server utilization, while limiting the connection bandwidth from infinity to 28 kbps increases the overhead by approximately 20%.

E. 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 7 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 [16][27].

F. Interrupt Coalescing

To study the effect of interrupt coalescing, we ran experiments with Apache and TUX under the SURGE workload with coalescing enabled. Figure 8 shows the cost breakdown with and without interrupt coalescing. The graphs are normalized to keep the CPU utilization at 50% for the base 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%.

G. Request Latency

We measured the effect of the emulated network parameters on the request latency. Due to the space constraints, we just summarize the results (see [31] for complete results).

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 increase the latency by almost factor of six. These data show that connection bandwidth limits have a direct effect on the TCP connection throughput. On the other hand, delay in the network affects the latency by introducing round-trip time delay for the connection open/close and a few initial packets of response that require acknowledgements from the other side before new data can be sent. The rest of the data are pipelined so delay does not have much effect.

VI. CONCLUSION

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, which confirms previously published results [13]. 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 [6][10] or doing processing at the socket event handler [16], would not introduce significant performance differences due to already low relative cost of such overhead under studied conditions. HTTP processing optimizations implemented in TUX made the cost of HTTP processing small comparatively to other component costs.

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] Apache Web Server. <http://www.apache.org>.
- [4] 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.
- [5] 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.
- [6] 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.
- [7] 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.
- [8] P. Barford and M. Crovella. Critical Path Analysis of TCP Transactions. In *Proceedings of ACM SIGCOMM Symposium on Communications Architectures and Protocols*, Aug. 2000.
- [9] M. Carson. NIST Net. <http://www.antd.nist.gov/nistnet>.
- [10] A. Chandra and D. Mosberger. Scalability for Linux Event-Dispatching Mechanisms. In *Proceedings of 2001 USENIX Annual Technical Conference*, June 2001.
- [11] B. Chen and R. Morris. Flexible Control of Parallelism in a Multiprocessor PC Router. In *Proceedings of 2001 USENIX Annual Technical Conference*, June 2001.
- [12] Gigaset, Inc. Gigaset cLAN Product Family. <http://www.gigaset.com/products>.
- [13] 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.
- [14] Hypertext Transfer Protocol 1.1 Specification. RFC2068. <http://www.w3.org>.
- [15] Infiniband Architecture Specification. Vol. 1&2. Rel. 1.0. http://www.infinibandta.org/download_spec10.html.
- [16] 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.
- [17] P. E. McKenney and K. F. Dove. Efficient Demultiplexing of Incoming TCP Packets. In *Proceedings of the SIGCOMM '92 Conference*, Aug. 1993.
- [18] 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.
- [19] 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.
- [20] 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.
- [21] Microsoft Corporation. Installation and Performance Tuning of Microsoft Scalable Web Cache (SWC 2.0). <http://www.microsoft.com/technet/iis/swc2.asp>.
- [22] J. C. Mogul. Operating System Support for Busy Internet Server. Technical Report Technical Note TN-49, Digital Western Research Laboratory, May 1995.
- [23] 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.
- [24] J. C. Mogul. The Case for Persistent-connection HTTP. In *Proceedings of ACM SIGCOMM'95 Conference*, Oct. 1995.
- [25] J. C. Mogul and K. K. Ramakrishnan. Eliminating Receive Livelock in an Interrupt-driven Kernel. *ACM Transactions on Computer Systems*, 15(3), Aug. 1997.
- [26] 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.
- [27] 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.
- [28] 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.
- [29] V. V. Panteleenko. Instantaneous Offloading of Web Server Load. Ph.D. Dissertation, Department of Computer Science and Engineering, University of Notre Dame, 2002.
- [30] 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.
- [31] V. V. Panteleenko and V. W. Freeh. Web Server Performance in a WAN Environment. Technical Report TR-02-02, University of Notre Dame, 2002.
- [32] L. Rizzo. Dummynet: A Simple Approach to the Evaluation of Network Protocols. In *Computer Communication Revue*, 27(2), Feb. 1997.
- [33] 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.
- [34] 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.
- [35] The Standard Performance Evaluation Corporation. SPECWeb96/SPECWeb99. <http://www.spec.org/osg/>.
- [36] TUX Web Server 2.0. <http://www.redhat.com/products/software/tux>.
- [37] A. van de Ven. kHTTPd Linux HTTP accelerator. <http://www.fenrus.demon.nl>.