MAY 1995

WRL
Technical Note TN-49

Operating Systems Support for Busy Internet Servers

Jeffrey C. Mogul

d|i|g|i|t|a|l **Western Research Laboratory** 250 University Avenue Palo Alto, California 94301 USA

Operating Systems Support for Busy Internet Servers

Jeffrey C. Mogul

May, 1995

Abstract

The Internet has experienced exponential growth in the use of the World-Wide Web, and rapid growth in the use of other Internet services such as USENET news and electronic mail. These applications qualitatively differ from other network applications in the stresses they impose on busy server systems. Unlike traditional distributed systems, Internet servers must cope with huge user communities, short interactions, and long network latencies. Such servers require different kinds of operating system features to manage their resources effectively.

This Technical Note is an expanded version of a paper that was inadvertently omitted from the *Proceedings of the Fifth Workshop on Hot Topics in Operating Systems (HotOS-V)*. The correct citation for that paper is:

Jeffrey C. Mogul. Operating System Support for Busy Internet Servers. In *Proc. HotOS-V*, pp. addendum. Orcas Island, Washington, May, 1995.

digital **Western Research Laboratory** 250 University Avenue Palo Alto, California 94301 USA

Table of Contents

List of Figures

1. Introduction

The uses to which we put computers have evolved over time, starting with simple calculation, then adding database management, and finally communication. While much of the original research in computer networking was aimed at promoting wide-area resource sharing, until recently our networks only supported relatively local resource-sharing (files, databases, printers, and perhaps CPU cycles). Wide-area networks (WANs) were used mostly for email and remote terminal access, with only limited successes in other applications.

In 1993, this changed. The phrase ‘‘exponential growth’’ is often misused, but it is accurate when applied to the World-Wide Web. The Web has succeeded because it makes it easy for people to share information, the most valuable resource of a technological society (especially now that storage and CPU cycles are so cheap). With this success and growth has come a number of scaling problems. The networking research community has recognized this for a long time, and has produced a broad literature addressing the situation, as well as some lively debates over appropriate solutions.

In contrast, the operating systems research community has largely ignored these particular scaling issues. We have concentrated instead on scaling in relatively simple distributed systems, which share traditional hardware resources (cycles, memory, and files) and which mostly run in local area networks (LANs) of mutually cooperating systems. Those research efforts, such as AFS [6], that did look at wide-area issues still tended to treat the problem as one of extending the LAN model.

But Internet services are qualitatively different from distributed systems, and create different scaling problems. Organizations operating busy Internet Information Server (IIS) systems have begun to discover this, have run into these scaling problem, and need guidance and solutions from the operating systems community. In this paper, I cover some of the issues that arise when one tries to operate a high-volume IIS system, and I will suggest some possible solutions and areas for further research.

Many of the observations in this paper were inspired by our experience, at Digital’s Palo Alto research laboratories, in running several large IIS systems. We have been running Digital’s main Internet gateway for over a decade, starting with email and USENET, then adding a popular FTP archive, and finally a busy Web server. Most recently, in collaboration with the State of California, we built and operated a Web server to provide timely information about the 1994 statewide elections. This server handled over a million HTTP and Gopher requests in a single 24-hour period [12], and taught us a lot about how real-world loads differ from simulated loads. Figure 1 shows hourly samples of HTTP request rates for this service.

2. Characteristics of popular services

The most popular server-based Internet information services are the World-Wide Web, electronic mail (SMTP [16]), USENET news (NNTP [8]), and FTP [17] archive services. Many people expect server-based electronic commerce to become popular in the next few years, but it is not yet clear what technology will be employed. Each of these protocols has its own characteristics. In this paper, for reasons of space, I will focus on the World-Wide Web, although

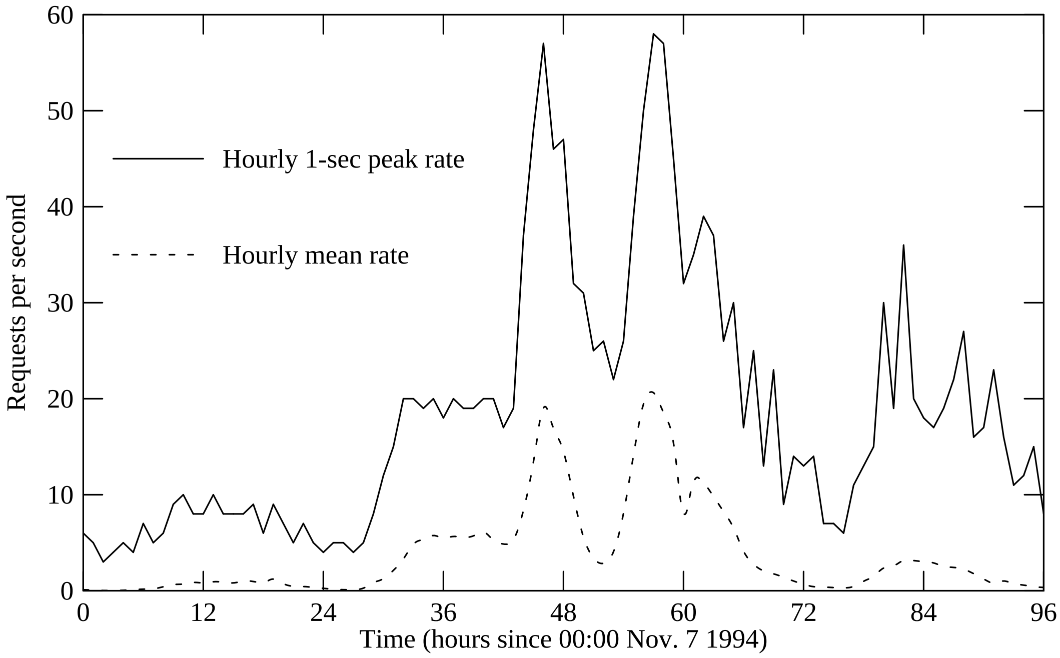


Figure 1: HTTP request rates for election server

many of my comments apply to all of these services. I am not considering continuous media services (realtime audio and video) because these are qualitatively and quantitatively distinct. And I am not addressing the problems of ''Internet Service Providers,'' which provide packet-level or login-level Internet access.

The Web is composed primarily of servers for the Hypertext Transport Protocol [2] (HTTP) and for Gopher [1]. These two protocols are qualitatively similar: a client establishes a TCP connection to a server, sends a request message, receives a response message, and closes the connection.

The popular Internet information services share several characteristics that distinguish them from traditional ''distributed systems'' applications:

- **Huge ''user'' set**: A typical distributed application serves just one user, or maybe a small workgroup; in rare cases, it might serve a large organization with constrained membership. Internet information services tend to be available to all comers. This year, that means several million people; in a decade, it may mean hundreds of millions. These users are anonymous, naive, and certainly not trustworthy.
- **Short TCP connections**: With the exception of NNTP, these protocols create a new TCP connection for almost every request. This is bad for performance [14, 13], and should be changed, but for now is a fact of life.
- **Long and variable network delays**: Many Internet users cannot afford high-bandwidth, low-latency connections (or they may be physically far from a server, thus experiencing speed-of-light delays). The Internet may also experience congestion in places, which causes queueing delays or packet losses. TCP turns packet losses into increased latency, so the net result is similar.
- **Frequent network partitions**: Many Internet users connect via temporary dialup links, or patronize flaky service providers. We have observed frequent transient or permanent network partitions, as a result.

- **No single administrative domain**: Unlike a traditional distributed system, which can be managed by one authority, neither the Internet nor its users are subject to any specific authority. In particular, this means that there is no common authentication or protection mechanism, and thus it is essentially impossible to prevent some people from using a server while allowing others to use it.
- **Different penalties for failure**: When a distributed system fails or runs slowly, users complain to the management. When an IIS system fails or runs slowly, users form bad opinions of the server and stop using it. This both makes it more difficult to detect problems, and more important to prevent or quickly solve them.
- **No scheduled downtime**: A workgroup or company can tell its internal users that a system must be taken offline, for repair, maintenance, or upgrade. It may not be possible to do this with a server potentially used by millions of anonymous people.

All of these characteristics make it harder to manage an IIS system than a traditional distributed system, and lead to more extreme scaling issues.

While an IIS system has many of the characteristics of a timesharing system, including a large user community and many competing activities, it differs in several ways. Timesharing systems typically run a small set of long-lived processes for each user; IIS systems typically service many short-duration requests per user. In this respect, IIS systems somewhat resemble transaction-processing systems, although they do not always require frequent, fast, and synchronized updates of stable storage.

3. Lack of useful benchmarks

We face a complete lack of guidance in sizing IIS systems. We do not know how servers respond to heavy loads, in large part because there are no standard, well-designed benchmarks.

For example, how many requests per second can an HTTP server handle? More precisely, how does the server's response time vary with increasing load? Following the LADDIS NFS benchmark [9], we could define a server's maximum rate as the point where its response time rises above an arbitrary value (LADDIS uses 50 msec).

We must also clarify what a standard ‘‘request’’ is. LADDIS specifies a mix of various NFS operations and sizes, meant to reflect a typical load. Such an approach should also work for HTTP, but the LADDIS group spent months debating the proper mix; for HTTP, we will need to do extensive trace analysis of popular servers before we can reach a consensus.

Benchmarks such as LADDIS usually assume that clients will submit legal requests; they measure only ‘‘correct’’ operation. But ‘‘correct’’ operation is not ‘‘normal’’ operation; clients of Internet information services often make errors. (The 1994 California Election server saw error rates as high as 20 errors per second.) And since servers often emit extra logging information for erroneous requests, performance in ‘‘normal’’ operation may be much lower than performance in ‘‘correct’’ operation. We also discovered the hard way that when clients abort their TCP connections prematurely, this can trigger lurking server bugs that really hurt performance.

Because Internet clients are often buggy, and have poor mechanisms for reporting problems, we need to test performance in all sorts of failure scenarios. In real life, if a system has millions of users, some of them will always be causing trouble.

I also suspect that similar issues will arise in benchmarking mail servers, although here total message throughput per day is more important than response time. Mail server performance may depend even more on the distinction between ‘‘correct’’ and ‘‘normal’’ operation. For example, a server may limit the number of simultaneous outbound connections it is willing to create. As the number of non-responsive destinations increases, the server’s throughput may drop because most of its outbound resources are tied up waiting for timeouts. And as the queue of deferred transmissions grows, the server may spend an increasing portion of its time scanning this queue.

People developing software for IIS systems also need unusual ‘‘micro-benchmark’’ information about the host systems they are using. For example, should a UNIX® HTTP server create a new process (i.e., *fork*) for each request, or should it try to manage simultaneous requests in one process (using *select* to avoid blocking I/O)? It may be that this decision has little effect on execution speed, and should be made on other bases, but it is not easy to discover what *fork* costs, or how *select* performance scales with a large number of file descriptors. Since traditional distributed systems do not create new connections or contexts as often as a busy IIS system, the operating systems features whose performance has been carefully measured and optimized (context-switch time, RPC time, signal-handling time) may not be terribly relevant for IIS systems.

4. OS behavior under overload

We have no way to control the load offered to an IIS system, because we have no control over the number of clients, or over their aggressiveness. While TCP flow-controls the data transmitted over an existing connection, it provides no means to control the rate of requests for new connections, and the popular Internet information services use a new connection for each transaction. All an overloaded server can do is to reject new connection attempts, which results in disgruntled users.

We want our IIS systems to handle as high a request rate as possible, but when that rate is exceeded, we do not want them to suffer from congestive collapse or ‘‘livelock.’’ A livelocked server spends all of its resources on non-productive operations, such as rejecting new connections or aborting partially-completed ones (perhaps because they time out or lack sufficient memory resources), while failing to process any tasks to completion. In such a case, system throughput drops to zero. A well-engineered operating system should continue to complete tasks at or near its maximum throughput, even when the offered load increases without bound, but getting this right can be tricky [18].

Since a busy server can have many connections in progress (and the TCP specification requires that the server maintain a ‘‘TIME_WAIT’’ connection record for four minutes after the connection terminates [15]), managing memory resources may be more important than managing CPU cycles. Each TCP connection, for example, normally commits the system to a minimum amount of receive buffer space. If the system overcommits its buffer space, then it may have to discard arriving packets for want of a place to put them. However, it is rather unlikely that all active clients will transmit requests at once, and so it may be reasonable to overcommit, risking the possibility of packet loss in exchange for allowing a larger number of active connections. The system may also have to aggressively terminate stalled connections, which occupy valuable buffer space, although this risks dissatisfying clients with slow network connections.

5. OS performance and capacity scaling

Many traditional operating system facilities do not scale well to large numbers of active connections, in part because kernel implementors have been reluctant to use complex data structures or data compression techniques. If you face benchmarks measuring the performance of just a few connections, or a few active processes, you are better off using linear searches, rather than (say) balanced trees, which may not perform as well until they contain hundreds or thousands of elements.

For example, in TCP implementations derived from 4.2BSD, each connection uses a protocol control block (PCB). Most such systems use a linear search through a linked list of PCBs; this may be augmented with a single-entry cache of the last successful lookup. This approach scales quite badly for a large number N of active connections, since the cache is useless and the linear searches result in $O(N^2)$ behavior. McKenney and Dove describe the use of a hash-based PCB table [10], which works a lot better. However, it may not be the optimal solution for an IIS system, whose PCB table will contain mostly TIME_WAIT entries that should almost never be the target of a successful lookup. Perhaps these should be stored in a separate structure, using a space-efficient representation, and optimized for lookup ‘‘misses’’ rather than lookup ‘‘hits.’’

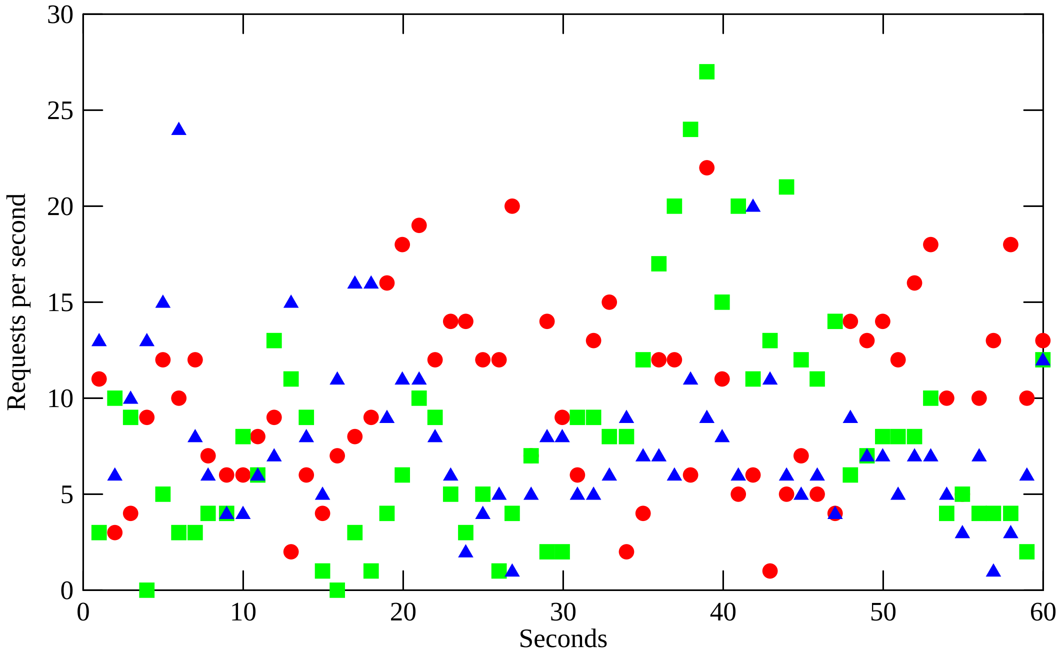
Similar issues affect the representation of the IP routing table. In current practice, an end-host’s routing table has an entry for each destination host, rather than an entry for each ‘‘destination network.’’ This is done because the topology of remote parts of the Internet is opaque to the local host, and the routing table may contain path-specific information (such as maximum packet size and round-trip time). 4.2BSD represented the routing table as a hash table with linearly-searched chains, but 4.3-Reno replaced this with a tree structure. This should provide reasonable time-efficiency for both lookups and table maintenance, but may not be as space-efficient (because of the storage used for internal nodes).

It may pay to aggregate entries for hosts with identical first-hop routes and path characteristics, in order to save space, although implementing this could be tricky. And, since routing table entries are ‘‘soft’’ state, it may pay to delete them as soon as a TCP connection enters the TIME_WAIT state, rather than waiting for the PCB record to expire.

5.1. Scaling server performance

Most Internet information services are easily parallelized, since (by protocol design) the individual transactions are completely independent. Successful parallelization means avoiding any unnecessary resource bottlenecks, such as network interfaces. Current practice in implementing the Domain Name System [11] (DNS) helps solve this problem. When a single server name is bound to several distinct hosts with equal preference values, DNS servers return the bindings in random order; this causes DNS clients (which generally try the bindings in the order returned) to spread their connections among the server hosts, and allows us to scale up capacity using independent replicated servers.

Such DNS-based load balancing does not work perfectly, because once a DNS client has retrieved a binding, it tends to use it for a long time. Also, not all DNS servers properly randomize the bindings they return. We observed that although we attempted to load-balance between three hosts for the California Election server, quite often one server ended up handling most of the load. During the 1000 busiest individual seconds, at least half of the time one server handled 45% or more of the requests. One server got stuck with 62% or more of the requests during 100 of those seconds, and in several cases, one server handled 88% of the load.



For the minute of 09:48, November 9, 1994

Figure 2: Short-term HTTP request rates broken down by server

Figure 2 shows how the DNS-based load balancing worked during one of the busier one-minute periods. Each mark reflects one second's worth of HTTP requests for one of the three servers; each mark shape is associated with a particular server. During any given second, one server usually handles far more requests than its peers. No single server consistently ''wins'' or ''loses'' for more than a few seconds.

Instead of using independent replicated servers, one could add CPUs to a multiprocessor (SMP) system. This should improve the load balancing between CPUs, could improve cache behavior, and does eliminate some of the management problems associated with independent replicated servers. SMP systems, however, may run into contention bottlenecks as transaction rates increase, and cannot economically be scaled beyond a certain point.

One might expect at least two kinds of contention on a busy SMP IIS system. Several kernel data structures, such as memory allocators, scheduling information, and protocol control blocks, must be protected by locks, and could result in lock contention. It may be possible to avoid some of this contention by pre-allocating fungible resources among the CPUs.

IIS server applications may also contend for access to local files, especially when disk access is required. This could result in significant queueing delays at the disks. RAID technology might not help, since many IIS transactions involve small files or small updates. Since files generally are not fungible, it may be necessary to balance disk load by moving hotspot files from one disk to another. One can also use a log-structured file system to reduce the overhead for logging information about each transaction.

Some IIS servers create lots of short-lived processes. Most operating system schedulers are designed to handle relatively small numbers of long-lived processes. Can such a scheduler do a good job of balancing many short-lived processes across multiple CPUs? And on both SMP and uniprocessor systems, can the scheduler handle these processes without excessive overhead? We will need further investigation to answer these questions.

6. Robustness and security

Organizations that run IIS systems want good performance, but more than that they require reliable and secure operation. Fault-tolerance techniques and good engineering practice, including techniques to avoid congestive collapse, solve most of the reliability problem, but may still leave a system vulnerable to denial-of-service attacks.

An IIS system intended to be open to all potential users runs the risk that some of these users, either through malice or error, will monopolize its resources. For example, we accidentally discovered a denial-of-service attack to which possibly all 4.2BSD-derived systems are vulnerable, and against which there is no perfect protection; we assume that other such attacks remain unrevealed. Generally, a server must use timeouts on all of its finite resources in order to protect against clients that fail to release them. The operating system must allow the server application to control all these timeouts, since only the server may know how to make the tradeoff between insufficient security and premature connection failure.

7. Operating system desiderata

Our experience with building and running busy IIS systems shows us that some modest help from the operating system could make life a lot easier. This help falls into three categories: direct control over kernel timeouts and resource limits, a feedback path to allow server applications to optimize resource use, and mechanisms to allow on-line control and diagnosis of a congested system.

7.1. Direct control of timeouts and resources

Operating systems implementors frequently must make assumptions about how large to make a table, or how soon to timeout an operation, in order to protect kernel resources from runaway processes. They make these decisions with some understanding of what constitutes ‘‘reasonable’’ behavior, but sometimes this understanding is wrong. Or, a decision may have been right at first, but over a few years system parameters change enough to make it obsolete. Inevitably, someone will find a legitimate need to exceed one of these limits.

Given this dilemma, an operating system implementation should allow a system manager to arbitrarily increase any quota, and increase or reduce any timeout within the limits set by protocol specifications. Wherever possible, the system should allow such changes without rebooting. The system should also count the number of times that each quota or timeout is exceeded, and make these counts available to the system manager. And, of course, all these parameters should be properly documented.

No system has unlimited resources, and a system manager who has the ability to change limits also has the responsibility to live within the available means. The ‘‘cache kernel’’ model adopted by the V++ system [4] may ease this problem, by virtualizing as many kernel resources as possible. Even with this design it may be necessary to adjust the size of kernel resource caches, to optimize the cache-miss costs for a specific application.

7.2. Resource introspection

When an IIS system is dedicated to a specific server application, that application may want to closely manage the system’s resources. To do this, it may need to ‘‘introspect’’: to observe the global dynamic system behavior that results from many local activities. The server may be structured as a large number of independent processes, which must nevertheless coordinate their resource use.

For example, the server may want to use excess disk bandwidth and buffer cache pages to prefetch file data in anticipation of future requests [5]. But in order to do this without delaying higher-priority file operations, the server needs a mechanism (other than reading `/dev/kmem`) to discover the current disk queue length, and perhaps the buffer cache hit rate.

Similarly, a server could transmit unrequested ‘‘hint’’ information to a client, but should only do so if the network path to the client has sufficient bandwidth. This might require a mechanism allowing the server to inquire about network output queue lengths, round-trip times, or estimated bandwidths (kept by some protocol implementations, such as ‘‘TCP Vegas’’ [3]).

A server might also control use of system resources, such as memory or CPU cycles, by limiting the number of active connections. This could be done either by rejecting some new connections, or prematurely terminating extant connections that have exceeded some quota. To manage these resources, the server needs information about current usage: for example, short-term CPU load average, network buffering commitments, PCB table size, and process descriptor count.

Experience with network congestion control algorithms [7] has shown that even with relatively poor information about resource exhaustion, simple control algorithms implemented locally can prevent global congestion. We should be able to transfer this result to cooperating processes in an IIS system, but only if the system provides the necessary feedback.

7.3. Disaster management

Even the best-engineered IIS systems will occasionally suffer from congestive collapse; the system may contain a bug, or be improperly configured, or the network may be failing, or the system may be subject to a denial-of-service attack. We cannot prevent such episodes, so we need to be able to diagnose and repair them. In many cases, we may not be able to do this ‘‘off-line’’; diagnosis may require real-world loads, from clients whose behavior we cannot control.

This means that IIS systems must provide mechanisms that allow diagnosis and control even during a disaster. For example, some fraction of both CPU and I/O resources should be reserved for system management tasks. The system should provide means for obtaining snapshots of the state of any or all processes, and especially for learning their individual resource usage. We may need to know which resources are being overused, and which processes are doing the damage. We may also need to discover what kinds of errors or exceptional conditions a given process has encountered.

References

[1] F. Anklesaria, M. McCahill, P. Lindner, D. Johnson, D. Torrey, and B. Alberti. *The Internet Gopher Protocol (a distributed document search and retrieval protocol)*. RFC 1436, Internet Engineering Task Force, March, 1993.

[2] Tim Berners-Lee. *Hypertext Transfer Protocol (HTTP)*. Internet Draft draft-ietf-iiir-http-00.txt, IETF, November, 1993. This is a working draft.

[3] Lawrence S. Brakmo, Sean W. O'Malley, and Larry L. Peterson. TCP Vegas: New Techniques for Congestion Detection and Avoidance. In *Proc. SIGCOMM '94 Symposium on Communications Architectures and Protocols*, pages 24-35. London, August, 1994.

[4] David R. Cheriton and Kenneth J. Duda. A Caching Model of Operating System Kernel Functionality. In *Proc. First USENIX Symposium on Operating Systems Design and Implementation*, pages 179-193. Monterey, CA, November, 1994.

[5] James Griffioen and Randy Appleton. Reducing File System Latency using a Predictive Approach. In *Proc. 1994 Summer USENIX Conf.*, pages 197-207. Boston, MA, June, 1994.

[6] John H. Howard, Michael L. Kazar, Sherri G. Menees, David A. Nichols, M. Satyanarayanan, Robert N. Sidebotham, and Michael J. West. Scale and Performance in a Distributed File System. *ACM Transactions on Computer Systems* 6(1):51-81, February, 1988.

[7] Van Jacobson. Congestion Avoidance and Control. In *Proc. SIGCOMM '88 Symposium on Communications Architectures and Protocols*, pages 314-329. Stanford, CA, August, 1988.

[8] Brian Kantor and Phil Lapsley. *Network News Transfer Protocol*. RFC 977, Network Information Center, SRI International, February, 1986.

[9] Bruce E. Keith and Mark Wittle. LADDIS: The Next Generation in NFS File Server Benchmarking. In *Proc. Summer 1993 USENIX Conference*, pages 111-128. Cincinnati, OH, June, 1993.

[10] Paul E. McKenney and Ken F. Dove. Efficient Demultiplexing of Incoming TCP Packets. In *SIGCOMM92*, pages 269-279. Baltimore, MD, August, 1992.

[11] P. Mockapetris. *Domain Names - Concepts and Facilities*. RFC 1034, Network Information Center, SRI International, November, 1987.

[12] Jeffrey C. Mogul. *Network Behavior of a Busy Web Server and its Clients*. Research Report 95/5, Digital Equipment Corporation Western Research Laboratory, May, 1995.

[13] Jeffrey C. Mogul. *The Case for Persistent-Connection HTTP*. Research Report 95/4, Digital Equipment Corporation Western Research Laboratory, May, 1995. To appear in *Proc. SIGCOMM '95*.

[14] Venkata N. Padmanabhan and Jeffrey C. Mogul. Improving WWW Latency. In *Proc. Second WWW Conference '94: Mosaic and the Web*, pages 995-1005. Chicago, IL, October, 1994.

[15] Jon B. Postel. *Transmission Control Protocol*. RFC 793, Network Information Center, SRI International, September, 1981.

[16] Jonathan B. Postel. *Simple Mail Transfer Protocol*. RFC 821, Network Information Center, SRI International, August, 1982.

[17] J. Postel and J. Reynolds. *File Transfer Protocol (FTP)*. RFC 959, Network Information Center, SRI International, October, 1985.

[18] K. K. Ramakrishnan. Scheduling Issues for Interfacing to High Speed Networks. In *Proc. Globecom '92 IEEE Global Telecommunications Conf.*, pages 622-626. Orlando, FL, December, 1992.

WRL Research Reports

‘‘Titan System Manual.’’
Michael J. K. Nielsen.
WRL Research Report 86/1, September 1986.

‘‘Global Register Allocation at Link Time.’’
David W. Wall.
WRL Research Report 86/3, October 1986.

‘‘Optimal Finned Heat Sinks.’’
William R. Hamburgen.
WRL Research Report 86/4, October 1986.

‘‘The Mahler Experience: Using an Intermediate Language as the Machine Description.’’
David W. Wall and Michael L. Powell.
WRL Research Report 87/1, August 1987.

‘‘The Packet Filter: An Efficient Mechanism for User-level Network Code.’’
Jeffrey C. Mogul, Richard F. Rashid, Michael J. Accetta.
WRL Research Report 87/2, November 1987.

‘‘Fragmentation Considered Harmful.’’
Christopher A. Kent, Jeffrey C. Mogul.
WRL Research Report 87/3, December 1987.

‘‘Cache Coherence in Distributed Systems.’’
Christopher A. Kent.
WRL Research Report 87/4, December 1987.

‘‘Register Windows vs. Register Allocation.’’
David W. Wall.
WRL Research Report 87/5, December 1987.

‘‘Editing Graphical Objects Using Procedural Representations.’’
Paul J. Asente.
WRL Research Report 87/6, November 1987.

‘‘The USENET Cookbook: an Experiment in Electronic Publication.’’
Brian K. Reid.
WRL Research Report 87/7, December 1987.

‘‘MultiTitan: Four Architecture Papers.’’
Norman P. Jouppi, Jeremy Dion, David Boggs, Michael J. K. Nielsen.
WRL Research Report 87/8, April 1988.

‘‘Fast Printed Circuit Board Routing.’’
Jeremy Dion.
WRL Research Report 88/1, March 1988.

‘‘Compacting Garbage Collection with Ambiguous Roots.’’
Joel F. Bartlett.
WRL Research Report 88/2, February 1988.

‘‘The Experimental Literature of The Internet: An Annotated Bibliography.’’
Jeffrey C. Mogul.
WRL Research Report 88/3, August 1988.

‘‘Measured Capacity of an Ethernet: Myths and Reality.’’
David R. Boggs, Jeffrey C. Mogul, Christopher A. Kent.
WRL Research Report 88/4, September 1988.

‘‘Visa Protocols for Controlling Inter-Organizational Datagram Flow: Extended Description.’’
Deborah Estrin, Jeffrey C. Mogul, Gene Tsudik, Kamaljit Anand.
WRL Research Report 88/5, December 1988.

‘‘SCHEME->C A Portable Scheme-to-C Compiler.’’
Joel F. Bartlett.
WRL Research Report 89/1, January 1989.

‘‘Optimal Group Distribution in Carry-Skip Adders.’’
Silvio Turrini.
WRL Research Report 89/2, February 1989.

‘‘Precise Robotic Paste Dot Dispensing.’’
William R. Hamburgen.
WRL Research Report 89/3, February 1989.

‘‘Simple and Flexible Datagram Access Controls for Unix-based Gateways.’’
Jeffrey C. Mogul.
WRL Research Report 89/4, March 1989.

‘‘Spritely NFS: Implementation and Performance of Cache-Consistency Protocols.’’
V. Srinivasan and Jeffrey C. Mogul.
WRL Research Report 89/5, May 1989.

‘‘Available Instruction-Level Parallelism for Superscalar and Superpipelined Machines.’’
Norman P. Jouppi and David W. Wall.
WRL Research Report 89/7, July 1989.

‘‘A Unified Vector/Scalar Floating-Point Architecture.’’
Norman P. Jouppi, Jonathan Bertoni, and David W. Wall.
WRL Research Report 89/8, July 1989.

‘‘Architectural and Organizational Tradeoffs in the Design of the MultiTitan CPU.’’
Norman P. Jouppi.
WRL Research Report 89/9, July 1989.

‘‘Integration and Packaging Plateaus of Processor Performance.’’
Norman P. Jouppi.
WRL Research Report 89/10, July 1989.

‘‘A 20-MIPS Sustained 32-bit CMOS Microprocessor with High Ratio of Sustained to Peak Performance.’’
Norman P. Jouppi and Jeffrey Y. F. Tang.
WRL Research Report 89/11, July 1989.

‘‘The Distribution of Instruction-Level and Machine Parallelism and Its Effect on Performance.’’
Norman P. Jouppi.
WRL Research Report 89/13, July 1989.

‘‘Long Address Traces from RISC Machines: Generation and Analysis.’’
Anita Borg, R.E.Kessler, Georgia Lazana, and David W. Wall.
WRL Research Report 89/14, September 1989.

‘‘Link-Time Code Modification.’’
David W. Wall.
WRL Research Report 89/17, September 1989.

‘‘Noise Issues in the ECL Circuit Family.’’
Jeffrey Y.F. Tang and J. Leon Yang.
WRL Research Report 90/1, January 1990.

‘‘Efficient Generation of Test Patterns Using Boolean Satisfiablilty.’’
Tracy Larrabee.
WRL Research Report 90/2, February 1990.

‘‘Two Papers on Test Pattern Generation.’’
Tracy Larrabee.
WRL Research Report 90/3, March 1990.

‘‘Virtual Memory vs. The File System.’’
Michael N. Nelson.
WRL Research Report 90/4, March 1990.

‘‘Efficient Use of Workstations for Passive Monitoring of Local Area Networks.’’
Jeffrey C. Mogul.
WRL Research Report 90/5, July 1990.

‘‘A One-Dimensional Thermal Model for the VAX 9000 Multi Chip Units.’’
John S. Fitch.
WRL Research Report 90/6, July 1990.

‘‘1990 DECWRL/Livermore Magic Release.’’
Robert N. Mayo, Michael H. Arnold, Walter S. Scott, Don Stark, Gordon T. Hamachi.
WRL Research Report 90/7, September 1990.

‘‘Pool Boiling Enhancement Techniques for Water at Low Pressure.’’
Wade R. McGillis, John S. Fitch, William R. Hamburgen, Van P. Carey.
WRL Research Report 90/9, December 1990.

‘‘Writing Fast X Servers for Dumb Color Frame Buffers.’’
Joel McCormack.
WRL Research Report 91/1, February 1991.

‘‘A Simulation Based Study of TLB Performance.’’
J. Bradley Chen, Anita Borg, Norman P. Jouppi.
WRL Research Report 91/2, November 1991.

‘‘Analysis of Power Supply Networks in VLSI Circuits.’’
Don Stark.
WRL Research Report 91/3, April 1991.

‘‘TurboChannel T1 Adapter.’’
David Boggs.
WRL Research Report 91/4, April 1991.

‘‘Procedure Merging with Instruction Caches.’’
Scott McFarling.
WRL Research Report 91/5, March 1991.

‘‘Don’t Fidget with Widgets, Draw!.’’
Joel Bartlett.
WRL Research Report 91/6, May 1991.

‘‘Pool Boiling on Small Heat Dissipating Elements in Water at Subatmospheric Pressure.’’
Wade R. McGillis, John S. Fitch, William R. Hamburgen, Van P. Carey.
WRL Research Report 91/7, June 1991.

‘‘Incremental, Generational Mostly-Copying Garbage Collection in Uncooperative Environments.’’
G. May Yip.
WRL Research Report 91/8, June 1991.

‘‘Interleaved Fin Thermal Connectors for Multichip Modules.’’
William R. Hamburgen.
WRL Research Report 91/9, August 1991.

‘‘Experience with a Software-defined Machine Architecture.’’
David W. Wall.
WRL Research Report 91/10, August 1991.

‘‘Network Locality at the Scale of Processes.’’
Jeffrey C. Mogul.
WRL Research Report 91/11, November 1991.

‘‘Cache Write Policies and Performance.’’
Norman P. Jouppi.
WRL Research Report 91/12, December 1991.

‘‘Packaging a 150 W Bipolar ECL Microprocessor.’’
William R. Hamburgen, John S. Fitch.
WRL Research Report 92/1, March 1992.

‘‘Observing TCP Dynamics in Real Networks.’’
Jeffrey C. Mogul.
WRL Research Report 92/2, April 1992.

‘‘Systems for Late Code Modification.’’
David W. Wall.
WRL Research Report 92/3, May 1992.

‘‘Piecewise Linear Models for Switch-Level Simulation.’’
Russell Kao.
WRL Research Report 92/5, September 1992.

‘‘A Practical System for Intermodule Code Optimization at Link-Time.’’
Amitabh Srivastava and David W. Wall.
WRL Research Report 92/6, December 1992.

‘‘A Smart Frame Buffer.’’
Joel McCormack & Bob McNamara.
WRL Research Report 93/1, January 1993.

‘‘Recovery in Spritely NFS.’’
Jeffrey C. Mogul.
WRL Research Report 93/2, June 1993.

‘‘Tradeoffs in Two-Level On-Chip Caching.’’
Norman P. Jouppi & Steven J.E. Wilton.
WRL Research Report 93/3, October 1993.

‘‘Unreachable Procedures in Object-oriented Programing.’’
Amitabh Srivastava.
WRL Research Report 93/4, August 1993.

‘‘An Enhanced Access and Cycle Time Model for On-Chip Caches.’’
Steven J.E. Wilton and Norman P. Jouppi.
WRL Research Report 93/5, July 1994.

‘‘Limits of Instruction-Level Parallelism.’’
David W. Wall.
WRL Research Report 93/6, November 1993.

‘‘Fluoroelastomer Pressure Pad Design for Microelectronic Applications.’’
Alberto Makino, William R. Hamburgen, John S. Fitch.
WRL Research Report 93/7, November 1993.

‘‘A 300MHz 115W 32b Bipolar ECL Microprocessor.’’
Norman P. Jouppi, Patrick Boyle, Jeremy Dion, Mary Jo Doherty, Alan Eustace, Ramsey Haddad, Robert Mayo, Suresh Menon, Louis Monier, Don Stark, Silvio Turrini, Leon Yang, John Fitch, William Hamburgen, Russell Kao, and Richard Swan.
WRL Research Report 93/8, December 1993.

‘‘Link-Time Optimization of Address Calculation on a 64-bit Architecture.’’
Amitabh Srivastava, David W. Wall.
WRL Research Report 94/1, February 1994.

‘‘ATOM: A System for Building Customized Program Analysis Tools.’’
Amitabh Srivastava, Alan Eustace.
WRL Research Report 94/2, March 1994.

‘‘Complexity/Performance Tradeoffs with Non-Blocking Loads.’’
Keith I. Farkas, Norman P. Jouppi.
WRL Research Report 94/3, March 1994.

‘‘A Better Update Policy.’’
Jeffrey C. Mogul.
WRL Research Report 94/4, April 1994.

‘‘Boolean Matching for Full-Custom ECL Gates.’’
Robert N. Mayo, Herve Touati.
WRL Research Report 94/5, April 1994.

‘‘Software Methods for System Address Tracing: Implementation and Validation.’’
J. Bradley Chen, David W. Wall, and Anita Borg.
WRL Research Report 94/6, September 1994.

‘‘Performance Implications of Multiple Pointer Sizes.’’
Jeffrey C. Mogul, Joel F. Bartlett, Robert N. Mayo, and Amitabh Srivastava.
WRL Research Report 94/7, December 1994.

‘‘How Useful Are Non-blocking Loads, Stream Buffers, and Speculative Execution in Multiple Issue Processors?.’’
Keith I. Farkas, Norman P. Jouppi, and Paul Chow.
WRL Research Report 94/8, December 1994.

‘‘Recursive Layout Generation.’’
Louis M. Monier, Jeremy Dion.
WRL Research Report 95/2, March 1995.

‘‘Contour: A Tile-based Gridless Router.’’
Jeremy Dion, Louis M. Monier.
WRL Research Report 95/3, March 1995.

‘‘The Case for Persistent-Connection HTTP.’’
Jeffrey C. Mogul.
WRL Research Report 95/4, May 1995.

‘‘Network Behavior of a Busy Web Server and its Clients.’’
Jeffrey C. Mogul.
WRL Research Report 95/5, June 1995.

WRL Technical Notes

‘‘TCP/IP PrintServer: Print Server Protocol.’’
Brian K. Reid and Christopher A. Kent.
WRL Technical Note TN-4, September 1988.

‘‘TCP/IP PrintServer: Server Architecture and Implementation.’’
Christopher A. Kent.
WRL Technical Note TN-7, November 1988.

‘‘Smart Code, Stupid Memory: A Fast X Server for a Dumb Color Frame Buffer.’’
Joel McCormack.
WRL Technical Note TN-9, September 1989.

‘‘Why Aren’t Operating Systems Getting Faster As Fast As Hardware?’’
John Ousterhout.
WRL Technical Note TN-11, October 1989.

‘‘Mostly-Copying Garbage Collection Picks Up Generations and C++.’’
Joel F. Bartlett.
WRL Technical Note TN-12, October 1989.

‘‘The Effect of Context Switches on Cache Performance.’’
Jeffrey C. Mogul and Anita Borg.
WRL Technical Note TN-16, December 1990.

‘‘MTOOL: A Method For Detecting Memory Bottlenecks.’’
Aaron Goldberg and John Hennessy.
WRL Technical Note TN-17, December 1990.

‘‘Predicting Program Behavior Using Real or Estimated Profiles.’’
David W. Wall.
WRL Technical Note TN-18, December 1990.

‘‘Cache Replacement with Dynamic Exclusion’’
Scott McFarling.
WRL Technical Note TN-22, November 1991.

‘‘Boiling Binary Mixtures at Subatmospheric Pressures’’
Wade R. McGillis, John S. Fitch, William R. Hamburgen, Van P. Carey.
WRL Technical Note TN-23, January 1992.

‘‘A Comparison of Acoustic and Infrared Inspection Techniques for Die Attach’’
John S. Fitch.
WRL Technical Note TN-24, January 1992.

‘‘TurboChannel Versatec Adapter’’
David Boggs.
WRL Technical Note TN-26, January 1992.

‘‘A Recovery Protocol For Spritely NFS’’
Jeffrey C. Mogul.
WRL Technical Note TN-27, April 1992.

‘‘Electrical Evaluation Of The BIPS-0 Package’’
Patrick D. Boyle.
WRL Technical Note TN-29, July 1992.

‘‘Transparent Controls for Interactive Graphics’’
Joel F. Bartlett.
WRL Technical Note TN-30, July 1992.

‘‘Design Tools for BIPS-0’’
Jeremy Dion & Louis Monier.
WRL Technical Note TN-32, December 1992.

‘‘Link-Time Optimization of Address Calculation on a 64-Bit Architecture’’
Amitabh Srivastava and David W. Wall.
WRL Technical Note TN-35, June 1993.

‘‘Combining Branch Predictors’’
Scott McFarling.
WRL Technical Note TN-36, June 1993.

‘‘Boolean Matching for Full-Custom ECL Gates’’
Robert N. Mayo and Herve Touati.
WRL Technical Note TN-37, June 1993.

‘‘Ramonamap - An Example of Graphical Groupware’’
Joel F. Bartlett.
WRL Technical Note TN-43, December 1994.

‘‘Circuit and Process Directions for Low-Voltage Swing Submicron BiCMOS’’
Norman P. Jouppi, Suresh Menon, and Stefanos Sidiropoulos.
WRL Technical Note TN-45, March 1994.

‘‘Experience with a Wireless World Wide Web Client’’
Joel F. Bartlett.
WRL Technical Note TN-46, March 1995.

‘‘I/O Component Characterization for I/O Cache Designs’’
Kathy J. Richardson.
WRL Technical Note TN-47, April 1995.

‘‘Attribute caches’’
Kathy J. Richardson, Michael J. Flynn.
WRL Technical Note TN-48, April 1995.

‘‘Operating Systems Support for Busy Internet Servers’’
Jeffrey C. Mogul.
WRL Technical Note TN-49, May 1995.