OCTOBER 1995

WRL
Research Report 95/5

Network Behavior of a Busy Web Server and its Clients

Jeffrey C. Mogul

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

Network Behavior of a Busy Web Server and its Clients

Jeffrey C. Mogul

October, 1995

Abstract

The 1994 California Election server, which provided ''live'' election returns, handled over 1.5 million individual requests, including almost 1 million in a single 24-hour period. This may have been the most intensive single event on the Internet, to date, and so represented a novel experiment in how the network responds to heavy loads. We collected comprehensive traces and logs of server and network operation, allowing offline analysis of numerous statistics. This paper reports the results.

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

Table of Contents

List of Figures

List of Tables

1. Introduction

On November 9, 1994, the United States held mid-term congressional elections. The results of these elections were of acute interest to many voters, since they led to turnover in the majority party of both houses of Congress. 1994 was also the first year in which many non-technical citizens had access to the Internet, and several organizations set up Internet servers to communicate campaign and election information to voters.

The most populous state, California, in conjunction with researchers from Digital Equipment Corporation, set up a server to provide voters and other interested users extensive pre-election information, and ''live'' election returns as they became available. This server attracted immense interest, handling over 1.5 million requests from over 20,000 hosts (and at least that many individual users).

This may have been the single most intensive event on the Internet, to date, so it represented a novel experiment in the behavior of the Internet and its users. While few current Internet servers continually experience this kind of load, we expect such bursty events to recur. We also expect the typical load on more quotidian servers to increase dramatically, as the user population explodes.

We took advantage of this opportunity by collecting comprehensive packet traces and server logs during the period of peak access rates. Using these traces and logs, we can do a broad variety of off-line analyses to obtain statistical information about the clients, servers, and network. We can also reconstruct the dynamics of individual connections or sets of connections.

This paper reports on the results of some of these analyses. We looked at server behavior, server access patterns, client behavior, network packet arrival patterns, and aspects of the paths that packets took through the Internet.

2. Overview of the Election Server

Approximately six weeks before the 1994 general election, a group of researchers from several of Digital's research labs arranged with the California Secretary of State's office to provide online election returns and pre-election voter information. The state would provide raw materials for the voter information, and a direct feed of returns once the polls had closed. Digital would operate both a World-Wide Web (WWW) server using the Hypertext Transfer Protocol (HTTP [2]), and a Gopher [1] server. Although the state officials seemed more interested in the Gopher server, we expected that the WWW server would prove far more popular, and concentrated our efforts there.

Because it took over a month for the final returns to be certified, and because much of the voter and return information may be of continued use, the server will continue to operate indefinitely. The WWW server may be reached as either of

```
http://www.election.ca.gov/
http://www.election.digital.com/
```

The Gopher server may be reached as either of

```
gopher://gopher.election.ca.gov/
gopher://gopher.election.digital.com/
```

You may find it easier to understand the content descriptions below if you first browse through the server.

2.1. Content

The content on the server is divided into static and dynamic pages. Static pages are those whose content does not vary with time, and include descriptions of ballot propositions and contested public offices, and statements provided by candidates for office and their political parties. Candidates are now allowed to provide photographs for the printed ballot pamphlet, and we made these available as well. We also obtained the official campaign finance statements, and somewhat laboriously transcribed them into an online form (this information was not otherwise easily available to voters). We also had almost all of this material translated into passable Spanish; we were unable to do translations into the other official languages (Chinese, Japanese, Tagalog, and Vietnamese). We of course provided elaborate hyperlinks between the various static pages.

Dynamic pages were generated at five-minute intervals from raw election return data provided by computers at the state's Teale Data Center. (After the first few days, updates came less frequently.) We generated several different kinds of Web pages from this data:

- **Bar graphs (by race or by county)**: For each race, a bar graph showed one line for each candidate, including the candidate's name, current vote percentage, and a horizontal bar proportional to the percentage. (For ballot propositions, we did not display bars.) Each bar was an ''inlined image,''[1] and so had to be retrieved separately from the server. We only generated bars for integral percentages, and we expected that client browsers would quickly build up a cache containing most of the necessary bar images. (See section 5 for a discussion of how well this worked.)
- **County maps**: For each race, a map showed how each of the 59 counties was voting (color-coded to show which candidate was leading in each county). We also generated per-county bar graphs, identical in format to the race-by-race bar graphs.
- **Television format**: six pages showing results for the major statewide races and propositions, formatted for display on an NTSC television screen. These were used by several cable television channels in lieu of generating their own graphics. Each page contained one dynamic and two static inlined images.

Since the images used in the bar-graph pages were themselves static, client caching of these caused no trouble. However, client caching of the county-map and TV-format images could have been a problem, since these images changed from time to time. We solved this by including a version number in the image file name, and changed the URLs in the enclosing HTML files whenever a new image was generated. We also created version-numbered instances of many of the dynamic HTML files, although not for those reached via county maps.

---

[1] *Inlined images* are files containing graphical elements, for insertion between or near textual elements.

We expected users to periodically re-request the HTML files for the dynamic pages, thus causing them to receive the most recent images as well. Many clients reach Web servers via caching relays or proxy servers [6]; these caching relays intercept the requests for HTML files and so can hide updates of the dynamic pages from clients. In most cases, properly informed users could work around this problem (for example, by hitting a ‘‘Reload’’ button). We discovered, however, that at least one major Internet service provider was caching accesses to our Gopher service, and was never updating its cache (so its users saw only the earliest returns). We suggest that implementors keep in mind that Web and Gopher content may not be static, and that designers of dynamic content provide ‘‘footnotes’’ instructing users how to work around caching relays.

The total content (including all versions of the dynamic pages) amounted to just 6981 Kbytes. This was less than 3% of the RAM on our server systems (see section 3.2), so we believe that almost all file reads were satisfied from the buffer cache. That is, almost no disk I/O had to be done to retrieve the content files.

2.2. Privacy

Although we kept extensive logs and traces of server and network activity, we recognized the need to protect the privacy of our users. We kept no logging information that directly identified individuals, and we will not reveal the names or addresses of client hosts, nor use them except to gather statistical information. We also have not released any statistics about the relative popularity of individual candidate, party, or result pages, since this could be used to gauge voter interest and perhaps could be used to design campaign strategies.

3. Hardware and software configuration

Prior to the election, we had no idea how many requests we would be receiving, but we made a wild guess that we might see a million requests (which turned out not to be far from the actual count). We realized that such a request rate could run up against several bottlenecks: Internet capacity, router throughput, LAN capacity, and server throughput. We also realized that if any part of the system failed, we would not have a second chance, so we wanted a highly redundant system.

3.1. Network configuration

Since we had 10 Mbit/sec (or faster) connections to the Internet via both AlterNet and BARRNet, a 10 Mbit/sec Ethernet LAN, and a router rated to run at full Ethernet rates, we did a crude calculation that convinced us that our local network infrastructure could handle the load. (Although we had a ‘‘Sniffer’’ monitoring the LAN, we neglected to log network utilization rates at regular intervals; spot checks showed one-second load averages peaking at about 70%, and long-term averages around 30%. See figure 3-3 for an approximation of the load averages.)

Our paired connections to the Internet came in handy, since during the day after the election (November 9th), which proved to be the day of heaviest load, it rained heavily and occasionally caused heavy packet losses on our microwave link to BARRNet. Once we detected the problem (it happened during lunch), we switched to a T3 wired connection, which eliminated the packet losses. We also constructed, but did not have to employ, a low-speed serial line connection to the Teale Data Center, for use in retrieving updated returns if the normal Internet path from Teale became congested by client traffic.

3.2. Server hardware

With sufficiently high-performance and redundant network connections in place, we turned our attention to the server systems. We used one three-processor and two dual-processor Digital 2100 model A500MP server machines; each processor was a Digital Alpha CPU rated at 110 SPECint92. Each dual-processor system was rated at 6,178 SPECrate_int92. The three-processor system had 512 MB of RAM; the dual-processor systems each had 256 MB. We maintained an identical copy of the server contents on the disks of each of the three server machines.

Since we wanted to expose only a single top-level URL, and hence a single host name, to clients, we used the Domain Name System (DNS) [13] CNAME mechanism to create a three-valued binding from `www.election.ca.gov` to the actual names of the three server hosts. We hoped that since modern DNS servers randomize the order in which they return the three bindings, clients would end up evenly balanced among the servers. Our experiences partially bore out this expectation; see section 4.3.

The use of several systems, instead of one large one, not only provided cost-effective performance scaling, it also provided a natural ‘‘warm spare’’ redundancy mechanism. We kept a fourth, somewhat slower, system running at all times, with its own copy of the database. If one of the main servers had failed, we would have rebooted the backup system after changing its IP address to match that of the failed system. We chose not to try to rebind the CNAME to the normal name (and address) of the backup system, since we had no idea how long it would take to propagate this change through the caches in the DNS.

3.3. Server software

The server machines ran DEC OSF/1 V3.0, which supports symmetric multiprocessing and thus made effective use of the dual processors. We used the NCSA `httpd` version 1.3 HTTP server, mostly because we had had extensive experience with this code and believed it could be trusted. However, we soon realized that this software might not support our estimated performance target, so I modified it slightly to avoid some inefficiencies.

Of course, modification introduced the possibility of new bugs, so we tested the new server both under a heavy artificial load, and on various production machines prior to election day, without finding any problems. Yet there was a bug, which only became visible shortly after the polls closed, as the request rate climbed dramatically. It turned out to be triggered when a client prematurely closed its end of an HTTP connection; the bug put the server process into a tight loop for several minutes (the NCSA server forks a new process for each connection). With increasing load, more users terminated their requests, causing more looping processes and eventually reducing the sustained request rate to a crawl.

It took me a while to locate the bug and prepare a fix, and anecdotal reports suggest that many potential users who tried the service during this period became frustrated and never came back. We suspect that we would have seen a much higher peak rate in the hours after the polls closed, had this bug not been present.

3.4. Log and trace collection

HTTP servers typically log some information about each request, but the original NCSA server does not log quite enough information to fully investigate the performance issues. In particular, it does not log connection duration, and it uses timestamps with 1-second resolution. I modified the server to record more extensive log information about each request, including connection duration, CPU time usage, and the number of actual disk reads done for each request. All timing information was done with approximately 1 msec resolution.

At fifteen minute intervals, we collected system-wide statistics on each server machine. These include system load averages, network interface statistics, a snapshot of the protocol control block (PCB) table, network buffer statistics, virtual memory statistics, and disk I/O statistics. (The disk I/O statistics, alas, proved useless because of a minor bug in the particular operating system release running on the servers, but we believe the file system cache was large enough to avoid almost all disk I/O.) We had intended to collect network and transport level statistics (using `netstat -s`), but neglected to do until approximately noon on the day after the election.

We also set up a workstation running the *tcpdump* program, to capture all of the traffic on the Ethernet. These traces covered all of November 8 (election day) and November 9, and most of the morning of November 10. We used *tcpdump* to capture the first 68 bytes of every packet, as well as the total packet length and a timestamp with microsecond resolution. Each set of one million packets was saved, without analysis, to a disk file and then compressed, yielding individual trace files of about 44 Mbytes. We ultimately traced about 209 million packet headers, requiring about 9 Gbytes of storage after compression; some of this data had to be stored offline, since we only had 6 Gb of spare disk space.

Promiscuous-mode passive monitoring, as done by *tcpdump*, has the advantage that it does not perturb the network being monitored, but one risks losing some packets because the monitor is overloaded. Fortunately, our monitor system (a DECstation 3000/400) was usually able to keep up. The software gave us an exact count of the number of dropped packets, which revealed a mean lost-packet rate of 0.011%. The peak lost-packet rate was 2012 per million packets, or 2%. (Figure 3-1 shows the actual discard rates, per million-packet set.) We may also have lost a few packets each time a new instance of *tcpdump* was started.

Figure 3-2 shows the packet capture rates over time; each sample is averaged over one million packets, so the peak rates were much higher. The peak capture rate reached one million packets in 545 seconds, a mean rate of about 1830 packets per second.

We can compute a lower bound on the Ethernet load average using the packet size and timestamp information in these traces; see figure 3-3. Each point represents the peak 1-sec load average seen during a 1-minute interval. Since tcpdump does not record Ethernet collisions, at higher rates this computation somewhat underestimates the actual load.

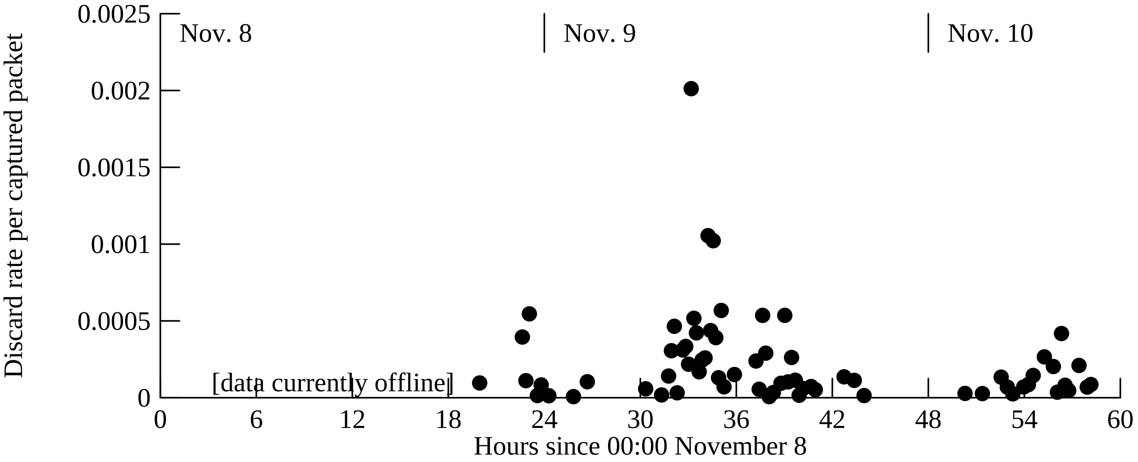


Figure 3-1: *tcpdump* discard rate vs. time

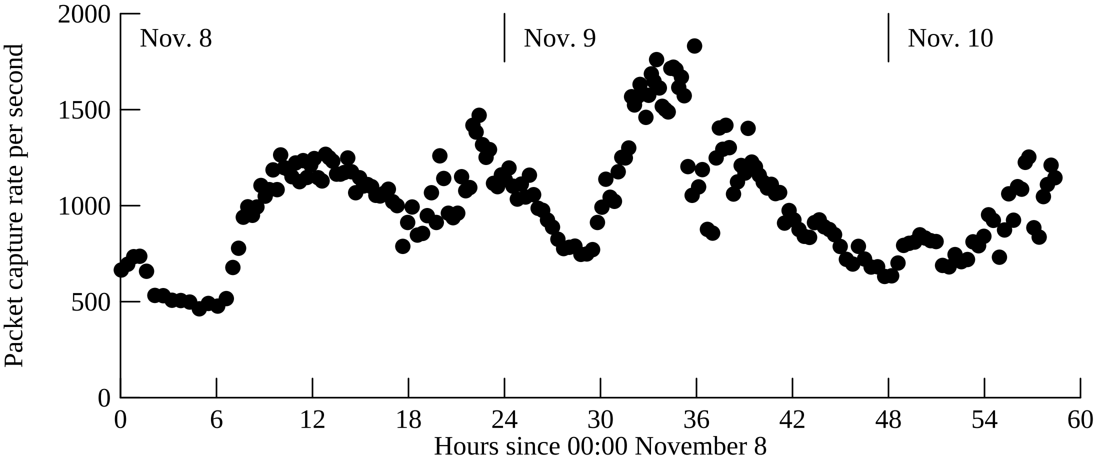


Figure 3-2: *tcpdump* capture rate vs. time

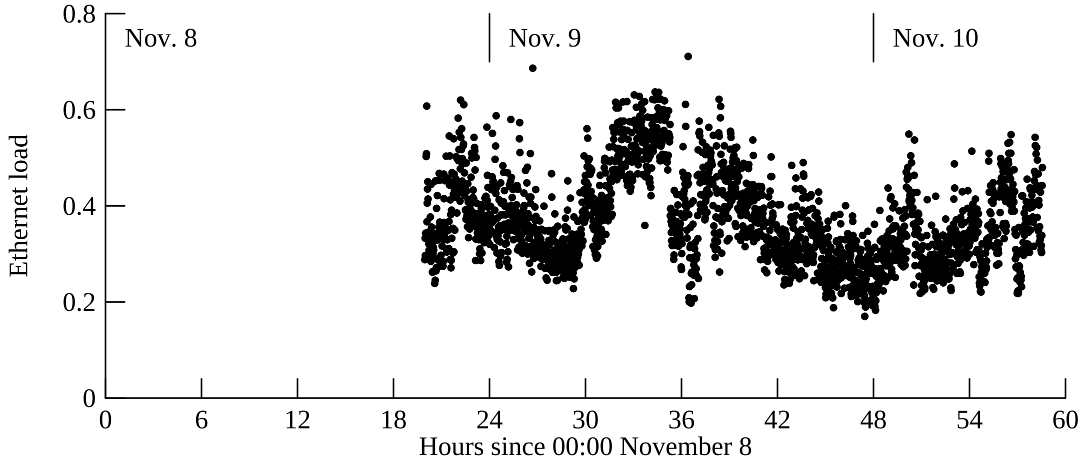


Figure 3-3: Lower bounds on Ethernet load average

Note that, because of the routing topology used for these servers, all packets sent by the servers appeared twice on the Ethernet (packets received by the servers appeared only once). This caused a somewhat higher Ethernet load than we would have seen with a more straightforward routing topology.

4. Overall server statistics

The most basic measure of server load is the rate at which clients issue requests. Figure 4-1 shows the hourly mean request rate (in requests per second) for about nine days around the time of the election. The dip at about noon on November 9 (hour 84) reflects the rain-induced network failure. Other dips reflect the wee hours of the morning, suggesting that most of our users were in or near our time zone.

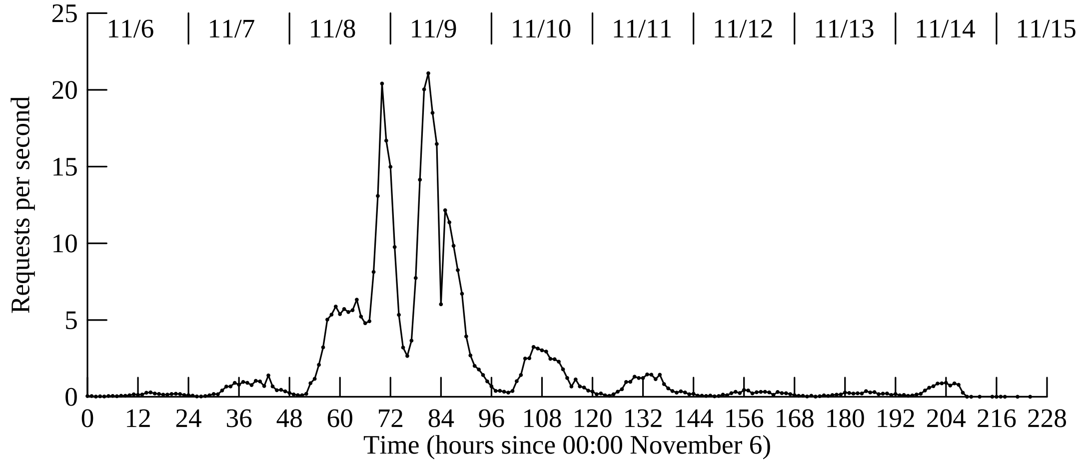


Figure 4-1: Request rates for all servers, average over 1-hour intervals

Peak request rates, over short time scales, far exceeded the hourly means. Figures 4-2 and 4-3 respectively show the peak 1-second and 1-minute rates for each hour. (Note that these rates reflect the times at which connections completed, not at which they were initiated.)

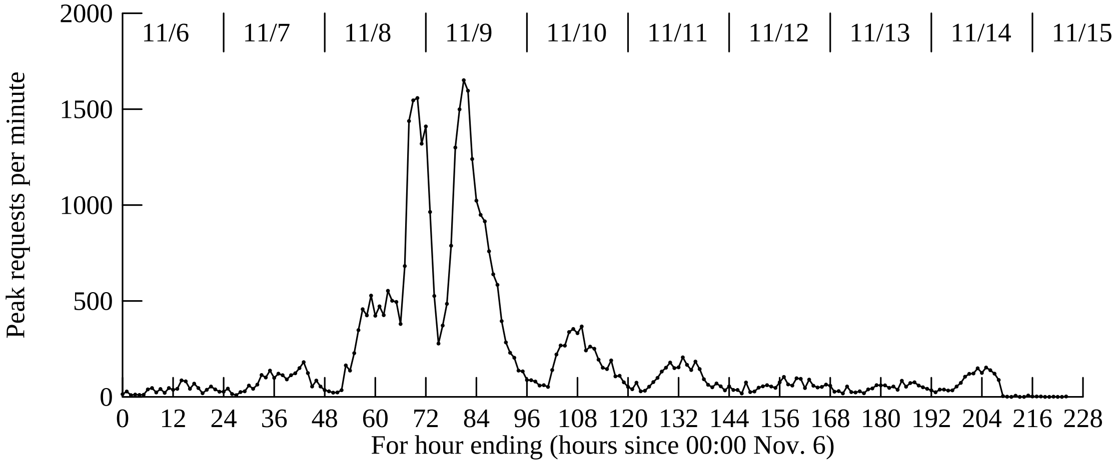


Figure 4-2: Peak one-minute request rate for all servers, reported hourly

Our logs contained accurate indications of connection duration, measured from the time that the *httpd* server process forked (this happens just after the TCP connection has been fully established, and so omits one round-trip time through the network) to the time that the connection has been successfully closed. Figure 4-4 shows the distribution of connection durations. Note that a significant number of connections lasted for many seconds, even though very few of the files transferred exceeded 10K bytes. This may either indicate lossy network connections, or attempts to retrieve large images over low-speed links.

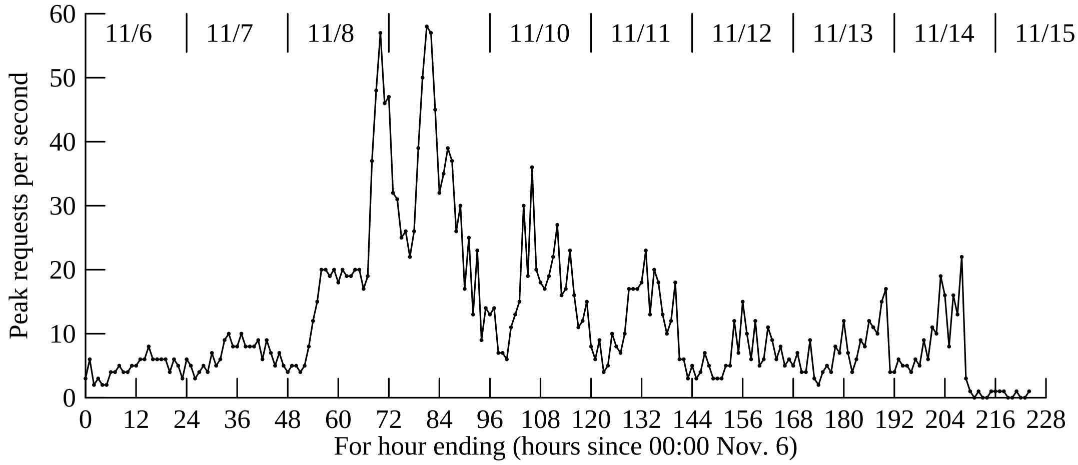


Figure 4-3: Peak one-second request rate for all servers, reported hourly

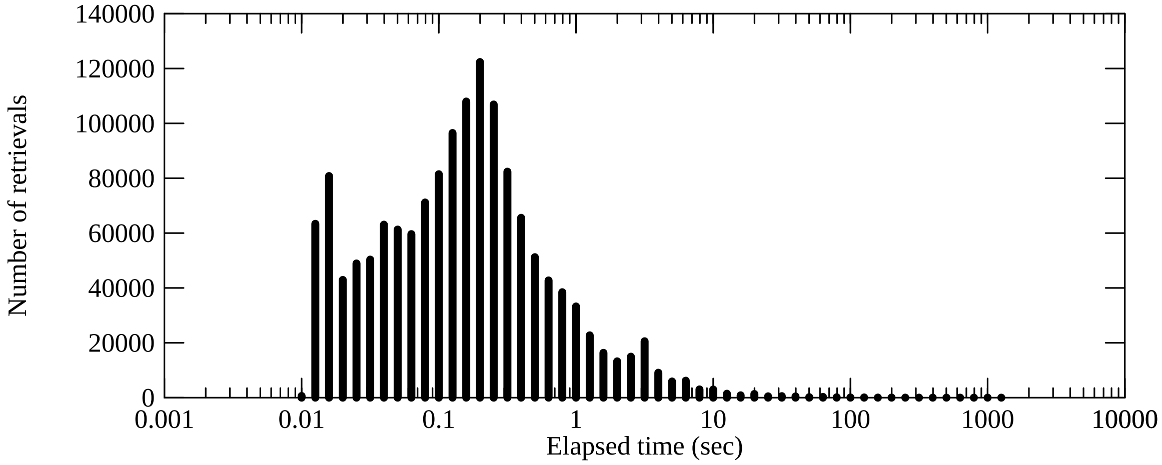


Figure 4-4: Distribution of connection durations

Figure 4-5 shows the distribution of the sizes, in bytes, of the files retrieved from the servers. (This does not include the considerable overhead data returned in an HTTP response.) Many retrievals fell into a narrow range of sizes under 100 bytes; these were all GIF-format images representing bars in the bar-graph pages. The mean retrieval size was 2394 bytes; the median was 958 bytes. (Ignoring 83406 zero-length retrievals, which are basically client cache validations, the mean was 2535 bytes and the median was 1025 bytes.)

Figure 4-6 shows how retrieval time (connection duration) correlates with the retrieved file size. The area of each dot on the graph is roughly proportional to the number of retrievals with a given size and duration, except that the lower limit on dot size is one pixel. This graph suggests that there is essentially no correlation between retrieval size and elapsed time for sizes below about 30K bytes. The overhead costs of creating TCP connections, creating server processes, exchanging and parsing request headers, and sending response headers overwhelm any per-byte costs for smaller files.

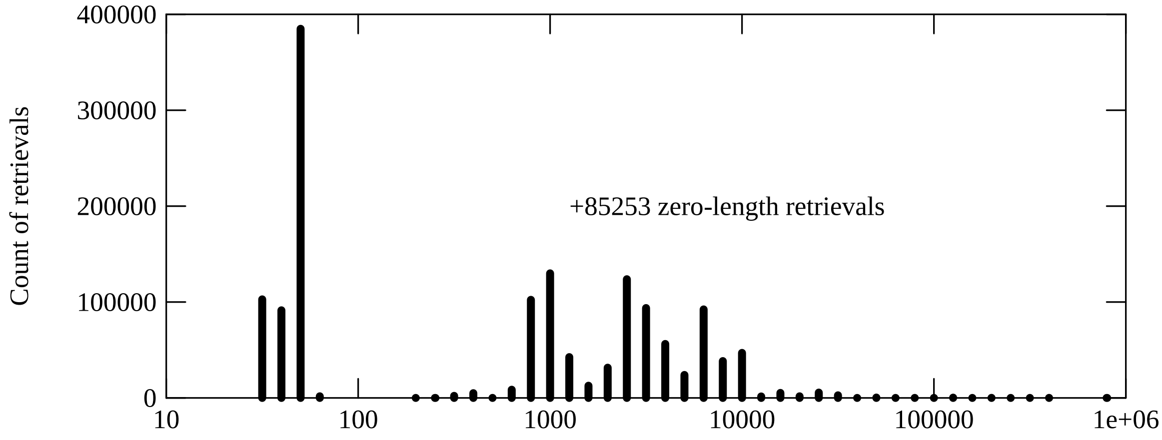


Figure 4-5: Distribution of retrieved file sizes

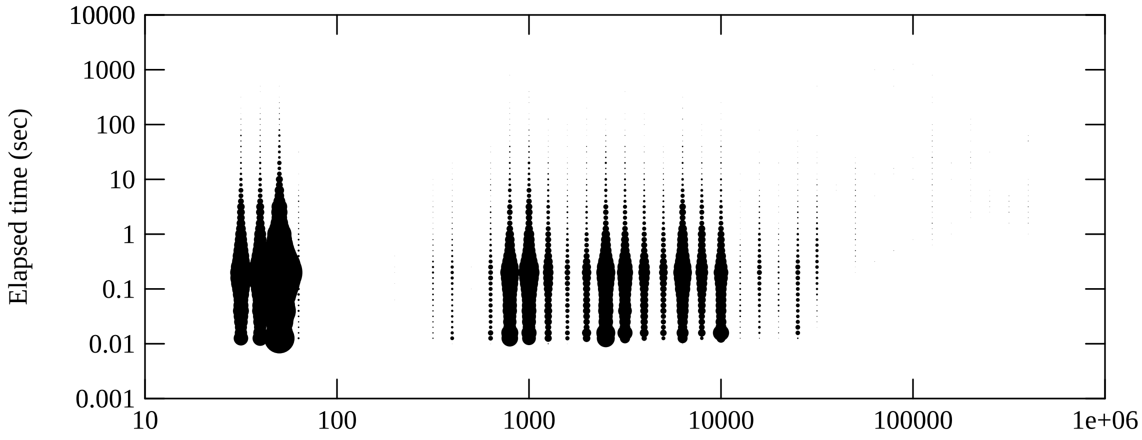


Figure 4-6: Correlation of connection duration with retrieved file size

4.1. Server resource usage

In addition to CPU time, an HTTP server uses several different kinds of system resources, including processes, TCP connections, TCP buffers, and file system buffers. These are all allocated from main memory. Although our server systems had more than enough RAM, we wondered just how much was necessary. That is, how did the servers make use of their memory resources?

We first looked at the number of entries in the protocol control block (PCB) table. Each TCP connection takes up an entry in this table. One would think that the table size should be approximately the number of active connections, but this is not so.

The TCP protocol specification [15] requires the host that closes a connection to remain in the TIME_WAIT state for a period of twice the maximum segment lifetime (2*MSL). This prevents a subsequent connection from accepting delayed duplicate packets from the closed connection. Since MSL should be two minutes, 2*MSL should be four minutes; however, DEC OSF/1 follows 4.3BSD practice and uses a one-minute timeout here.

The HTTP specification [2] requires that the server, not the client, close the TCP connection. This means that the average HTTP connection takes up a PCB table entry in the ESTABLISHED state for a few seconds, and then persists in the table in the TIME_WAIT state for quite a while longer. Hence, the number of TIME_WAIT entries can be far greater than the number of ESTABLISHED connections.

We recorded the contents of the PCB table every 15 minutes. Figure 4-7 shows how the breakdown by state varied over time, for all three servers taken together. (‘‘Opening’’ includes LISTEN, SYN_RCVD, and SYN_SENT; ‘‘Closing’’ includes CLOSE_WAIT, CLOSING, FIN_WAIT_1, FIN_WAIT_2, and LAST_ACK.) Note that nearly all of the table entries are for TIME_WAIT, and most of the rest are ‘‘Closing’’ states. Relatively few of the PCB table entries are used for actual live connections.

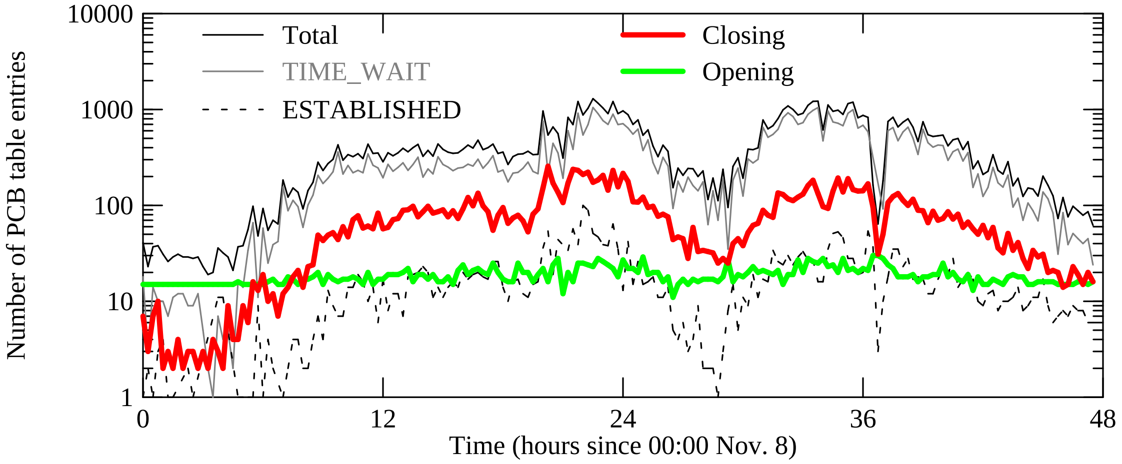


Figure 4-7: PCB Table statistics by TCP state

For all servers together, we recorded a peak PCB table size of 1297 entries. The peak number of TIME_WAIT entries was 1049, while the peak number of ESTABLISHED entries was 100. (The actual peaks might have been higher, since we sampled rather infrequently.) Note that if the operating system had used a four-minute timeout for the TIME_WAIT entries, this would have approximately quadrupled the number of these entries.

Table 4-1 shows the peak number of entries in each TCP state, as sampled at 15 minute intervals. The actual peaks may have occurred between samples. The table shows the peaks for each of the three servers, and for all three servers taken together (within the synchronization error of the sampling process). The latter may be smaller than the sum for all three servers, since the per-server peaks do not always coincide. The states, except for ESTABLISHED and TIME_WAIT, are categorized as ‘‘opening’’ or ‘‘closing’’ transient states, corresponding to the curves in figure 4-7.

The count of LISTEN states includes four other servers besides the HTTP server; the total number of HTTP listeners was always one per machine.

The number of SYN_RCVD states was limited to five by the *listen*() system used by the HTTP server. We now believe that this limit should be set much higher (perhaps in the hundreds or thousands), since connections can get stuck in this state if packets are being dropped in the network. With a limit of only five SYN_RCVD connections, the server can easily become ‘‘hung’’ for lengthy periods, during which it cannot accept any new connections. The rain-induced packet loss we experienced around noon on Nov. 9 did cause such a problem, which is

TCP State	Server A	Server B	Server C	All Servers
LISTEN[1]	5	5	5	15
SYN_RCVD[1]	5	5	5	15
SYN_SENT[1]	1	1	1	3
ESTABLISHED	40	35	26	100
CLOSE_WAIT[2]	46	113	48	163
CLOSING[2]	11	12	7	27
FIN_WAIT_1[2]	32	32	21	77
FIN_WAIT_2[2]	79	72	62	162
LAST_ACK[2]	5	7	14	22
TIME_WAIT	413	460	392	1049
All states	512	579	470	1297

Note 1: Connection is ‘‘opening’’
Note 2: Connection is ‘‘closing’’

Table 4-1: Peak counts of PCB table entries by TCP state

reflected as sharp dip in many of the graphs in this paper (for example, see figure 4-7, near hour 36).

Other sites have reported numerous connections stuck in the LAST_ACK state, probably because dialup clients became disconnected from the Internet before the server finished transmitting all of its buffered data. We found a few such connections, but never had a significant number at any one time. We also had a small number of connections stuck, some for periods of many hours, in FIN_WAIT_1, or less often in FIN_WAIT_2 (these were never more than a small fraction of the total).

The primary significance of these stuck connections (LAST_ACK and FIN_WAIT_*) is not that they take up PCB table space, but rather that they tie up buffer space for data that has been transmitted but not acknowledged. Over time, an HTTP server could lose significant amounts of kernel memory. This may require using a ‘‘keep alive’’ timer to garbage-collect such connections. How much memory did this table space require? Our logs also recorded memory-usage profiles every 15 minutes; figure 4-8 shows the results for three categories: ‘‘mbufs’’, used mostly to buffer packet headers and data; PCB table entries; and ‘‘sockets’’, used by the kernel to describe active connections. About half of the total use comes from mbufs; PCB table entries and sockets split the rest. In no case (that we sampled) did the total exceed 2 Mbytes, nor did the PCB table size ever exceed about 700 Kbytes. Therefore, we do not believe that even a very busy HTTP server requires much main memory.

However, large PCB tables not only consume memory; they also consume CPU cycles, because they must be searched on each packet arrival to find the corresponding TCP state record.

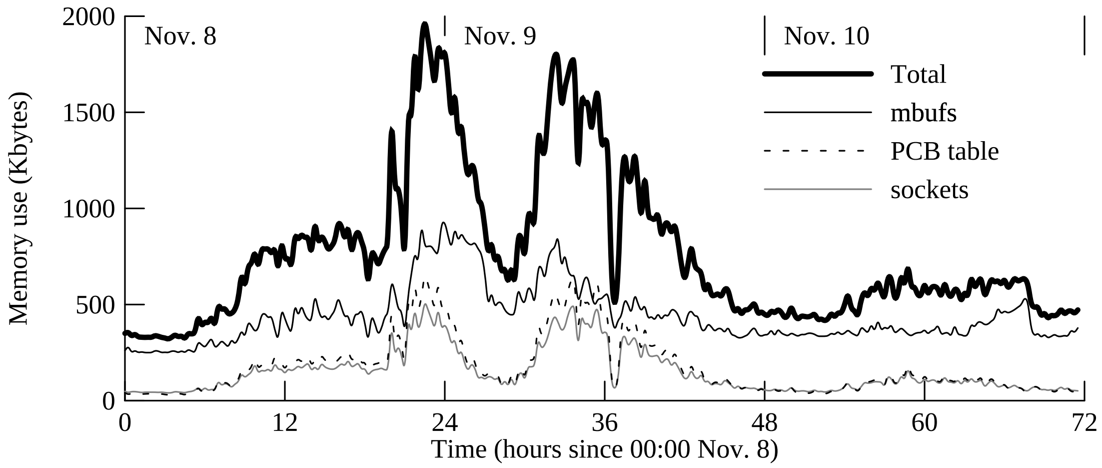


Figure 4-8: Memory used for network data structures and buffers

McKenney and Dove [11] have pointed out that the use of a linear list for this table can lead to poor performance, and suggest using a hash table for systems with large numbers of active connections. They point out that the use of small caches in front of a linear list do not work well in such applications.

In the case of an HTTP server, however, the number of active connections is small. Naive use of McKenney and Dove's hash-table structure would fill up with useless TIME_WAIT entries (these will almost never be the target of a lookup), slowing lookups and increasing the cost of table insertion and deletion.

A better solution might be to keep the TIME_WAIT entries in a separate structure, such as a queue (since entries will be removed in FIFO order). ''Useful'' PCB table entries could be kept in a hash-table structure. However, this may not be necessary for an HTTP server. I used the *tcpdump* traces to simulate *N*-entry LRU caches, and found reasonably good hit rates for small caches. These simulations were done using a *tcpdump* trace containing 127,367 packets arriving for the servers. This represents approximately 11 minutes starting at 9:40 AM on November 9, one of the peak load periods. Approximately 19198 connections were active during this interval, so these simulated caches warmed up rapidly.

Figure 4-9 shows the simulated hit rates for each individual server (dotted lines) and for all three servers taken as a single entity (solid line). For any individual server, a 16-entry cache would hit almost 80% of the time. If one server was handling the entire load, it would need a 64-entry cache to get an 80% hit rate. One could also look at these results as confirming that a moderately-sized hash table (containing entries only for active connections) would satisfy lookups in one or two comparisons.

4.2. Relative popularity of content files

Our logs contain the name of each file (or map hit) requested, which provides a breakdown of retrievals by file format. It also allows a division into static and dynamic content pages.

Figure 4-10 shows the mean retrieval rate (measured over two-hour periods, and plotted on a log scale) versus time, for both static and dynamic pages. Before the polls closed, most

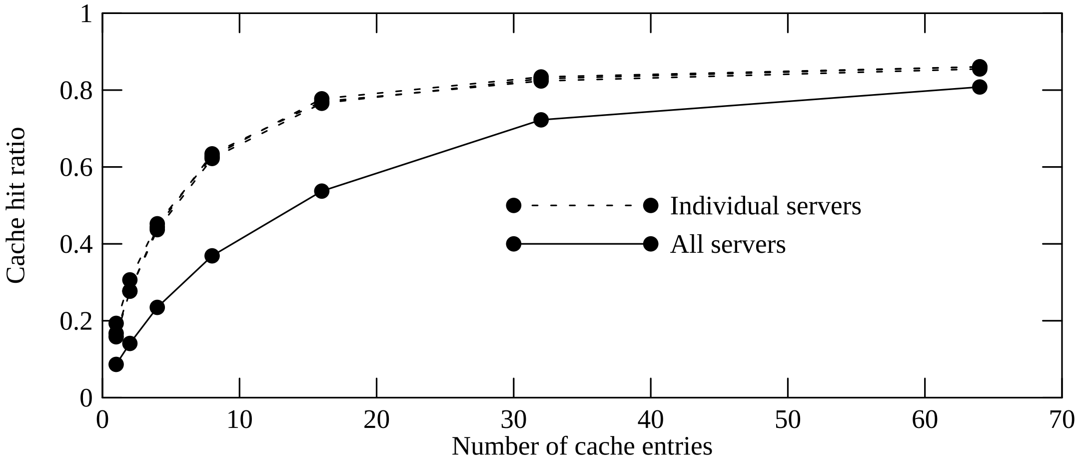


Figure 4-9: Simulated hit rates for small PCB lookup caches

retrievals (by an order of magnitude) were for static pages. After the polls closed (hour 60), dynamic pages led by a large margin, although not a factor of ten.

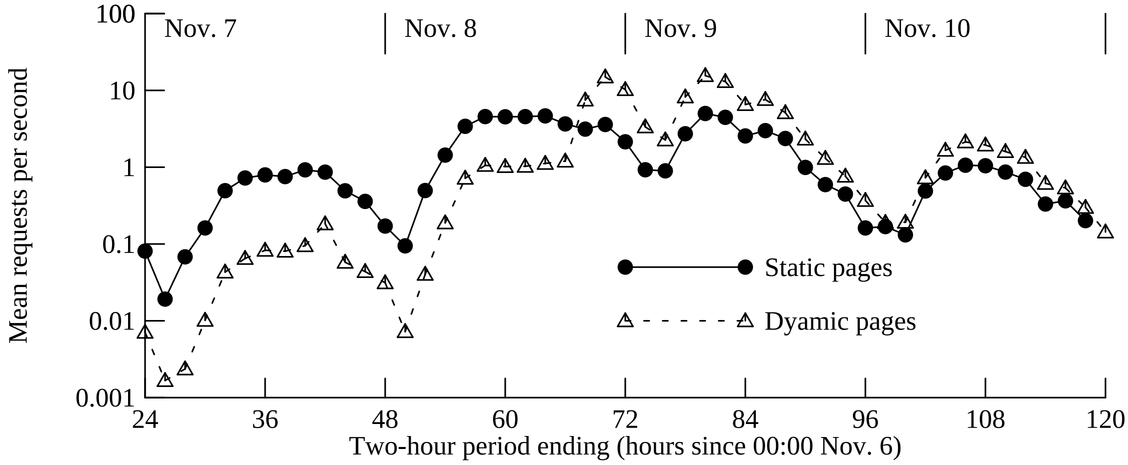


Figure 4-10: Relative popularity of static and dynamic pages

Figure 4-11 breaks down the mean retrieval rate (again, measured over two-hour periods) by file format. GIF and HTML files dominated all others by several orders of magnitude, with GIF files taking the lead after the polls closed (most of these were bar-graph elements).

Table 4-2 breaks down the retrievals by various different file name patterns. About 70% of the retrievals were GIF files. 62% of these were bar-graph elements; 11% were candidate photos; 6% were dynamically-constructed images such as maps of voting by counting; the remaining 21% were various static images such as banners, logos, and maps.

About 30% of the retrievals were HTML pages. 26% of these were for the three home pages (the multi-lingual master home page, and the English and Spanish home pages). Many users looked at the static HTML pages, such as candidate statements and descriptions of ballot propositions; these accounted for 18% of the HTML retrievals. 53% were for HTML pages containing election returns. The remaining HTML retrievals were for informational pages, such as a page with pointers to other election-related servers.

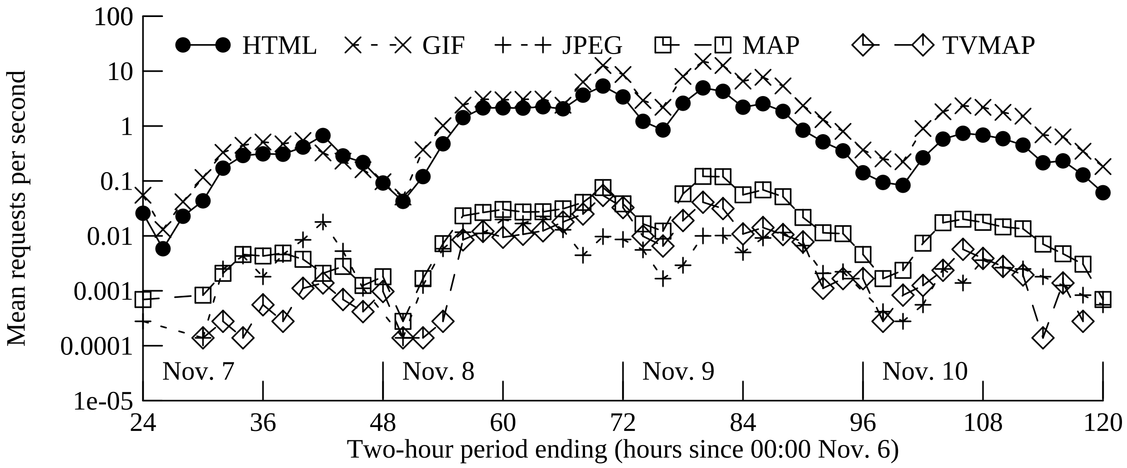


Figure 4-11: Relative popularity of different file formats

Only 1.3% of the HTML retrievals were for Spanish-language pages, although this may still represent several hundred users. We also had relatively few users of the pages formatted for television broadcast, although we do know that several cable channels used these extensively. Many users retrieved dynamically-constructed maps showing the breakdown of voting by county. Somewhat less often, a user clicked on one of these maps to see additional detail for a given county.

4.3. DNS-based load balancing effectiveness

In section 3.2 I described how we tried to use the Domain Name System (DNS) to spread out the request load among the three server machines. We knew that this would not work perfectly, since there are many caches in the DNS and these tend to reduce short-term randomness.

Over very long periods, the loads did nearly balance. Over the course of an entire day, the total number of requests handled varied among the servers by less than 10%. Figure 4-12 shows the mean hourly loads for each of the three servers, for election day and the day after; at this time scale, the loads are not entirely balanced, but generally are not far off.

At shorter time scales, however, the DNS-based load balancing clearly does not spread requests evenly. For example, figure 4-13 shows the per-second request rates for each of the three servers (denoted by different symbols) for one of the busiest minutes. The mean rate, over all three servers, is shown with a solid line. One can see that quite often, one of the servers carries far more than 1/3 of the total request rate. The total load varies enough from second to second to explain most of the inter-server variation; over the course of this minute, no server is obviously favored.

Figure 4-14 shows the cumulative distribution of load imbalance. This figure represents the busiest 10,000 one-second intervals (by total request rate), and it shows what fraction of the load was carried by the busiest server. (The curve looks essentially the same if one samples the 1000 or 100 busiest intervals.) For example, 25% of the time, the busiest server carried more than 54% of the total request rate. 5% of the time, the busiest server carried at least 2/3 of the request rate.

Pattern	Description	Count
/	Static Master home page	48984
/e/home.html	Static English home page	59546
/e/cand/.*.html	Static candidate statements	38083
/e/misc/.*.html	Static other stuff	47162
/e/other/.*.html	Static other stuff	512
/e/party/.*.html	Static party statements	5852
/e/prop/.*.html	Static proposition stuff	31152
/e/returns/.*.html	Dynamic returns html	219347
/e/returns/.*/page-tv..*.gif	Dynamic TV map pages	2521
/e/returns/.*/page..*.gif	Dynamic pages	59963
/e/returns/tv..*/page..*.gif	Dynamic TV non-map pages	4520
/s/home.html	Static Spanish home page	1487
/s/cand/.*.html	Static candidate statements/Spanish	805
/s/misc/.*.html	Static other stuff/Spanish	567
/s/other/.*.html	Static other stuff/Spanish	16
/s/party/.*.html	Static party statements/Spanish	119
/s/prop/.*.html	Static proposition stuff/Spanish	700
/s/returns/.*.html	Dynamic returns html/Spanish	1773
/s/returns/.*/page-tv..*.gif	Dynamic TV map pages/Spanish	25
/s/returns/.*/page..*.gif	Dynamic pages/Spanish	384
/s/returns/tv..*/page..*.gif	Dynamic TV non-map pages/Spanish	70
/s/.*	All Spanish pages	5961
/e/.*	All English pages	473071
/pics/bar/.*.gif	Dynamic bar graphs	615587
/pics/cand/.*.jpg	Static candidate JPEGS	1885
/pics/cand/.*.gif	Static candidate GIFs	114660
/pics/[^/]*.gif	Static misc pictures	207956
/.*ismap.*	Dynamic interactive map uses	8025
/.*tv-map.*	Dynamic TV interactive map	2760
/.*.html.*	All HTML	416113
/.*.gif.*	All GIF	998714
/.*.jpg.*	All JPEG	1885

Table 4-2: Distribution of requests by file type

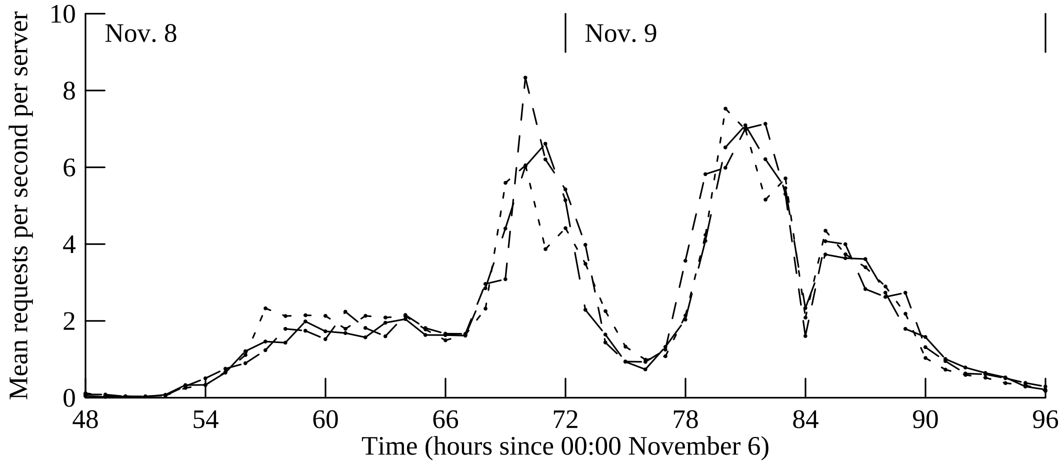
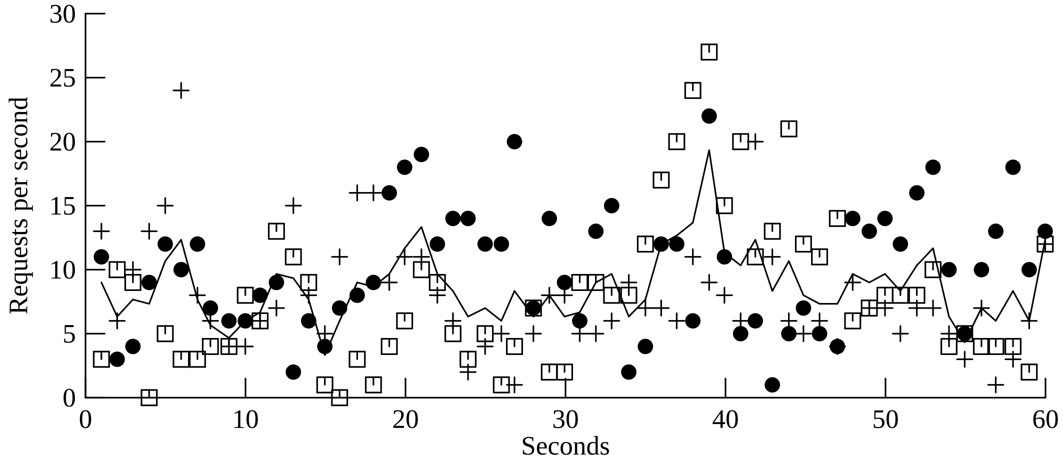


Figure 4-12: Mean request rates for each server, showing load balance



For the minute of 09:48, November 9

Figure 4-13: Instantaneous request rates for each server

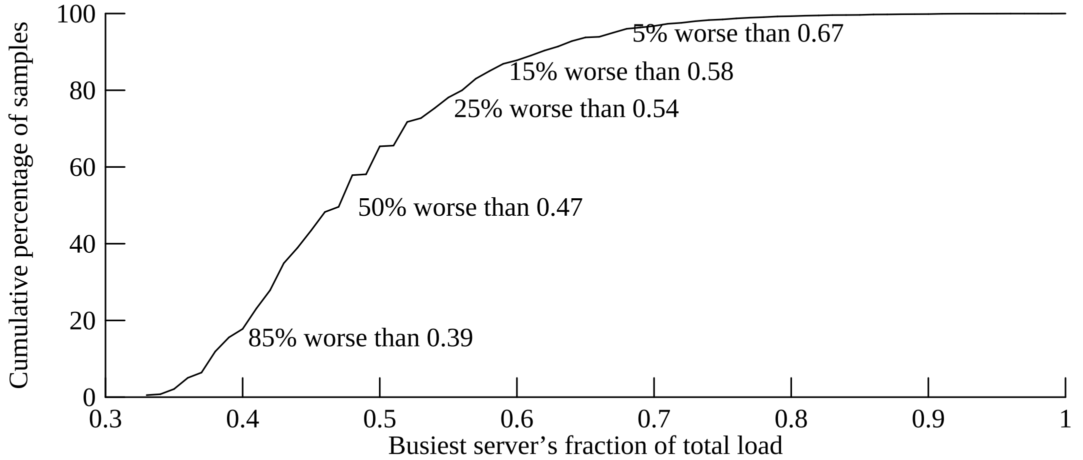


Figure 4-14: Cumulative distribution of load imbalances

Figure 4-15 shows load imbalances over longer measurement intervals.As the measurement interval increases, the likelihood of serious load imbalance decreases. For example, one server took more than half of the total load in 40% of the busiest 1-second intervals, but this happened in only 12% of the busiest 10-second intervals.

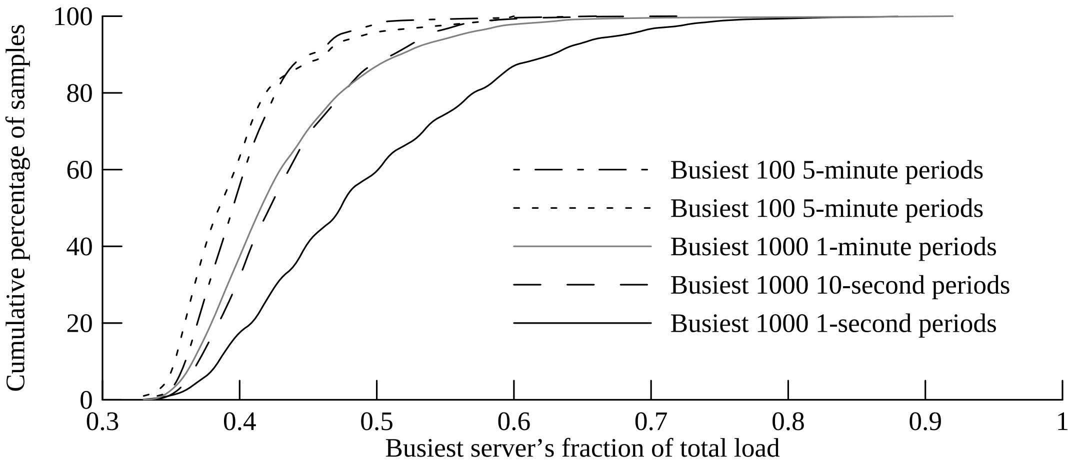


Figure 4-15: Cumulative distribution of longer-term load imbalances

Our goal in using multiple servers was both to provide redundancy and to improve performance. Our failure to balance the loads at the shortest time scale suggests that the DNS-based technique cannot provide linear scaling for server performance at peak request rates. Three servers may do better than one server, but probably not three times as well. The clients will most likely see slightly increased response times, not connection failures, since the second-by-second variation in request rate gives a temporarily overloaded server time to catch up.

5. Per-client statistics

How did a typical client use the Election Server? Is there, in fact, a ‘‘typical’’ client? Our logs included client host addresses, so we can analyze the per-client usage patterns. We looked at 1,494,003 retrievals starting on November 6.

Figure 5-1 shows the cumulative distribution of the number of requests per client host. The horizontal axis shows the number of retrievals per client; the vertical axis shows the cumulative number of retrievals. For a given *x* value, the corresponding *y* value shows the total number of retrievals done by client hosts that each did *x* retrievals or fewer.

An especially large number (1083) of hosts made exactly six requests. Analysis of the requested file names reveals why: the top-level URL (the only one widely publicized) contains five inlined images. Almost 900 of these six-retrieval hosts fetched all five of these images. 1070 of these hosts (98%) fetched the top-level home page. The remaining hosts were probably running browsers that do not use images (such as *Lynx*) or with image-retrieval disabled; these clients looked a little deeper into the server.

50% of the requests were done by client hosts that did 155 requests or fewer; 75% were done by clients that did 477 requests or fewer. This suggests that most client hosts were single-user machines (workstations or PCs).

However, a small set of hosts did many more retrievals than the ‘‘typical’’ host. While only 90 hosts (out of 21417) each did more than 1000 retrievals, these hosts accounted for almost 18% of the retrievals. 15 hosts each did more than 3000 retrievals, but accounted for 10% of the total. And 4 hosts did more than 10,000 retrievals (one did 33667), accounting for 6% of the

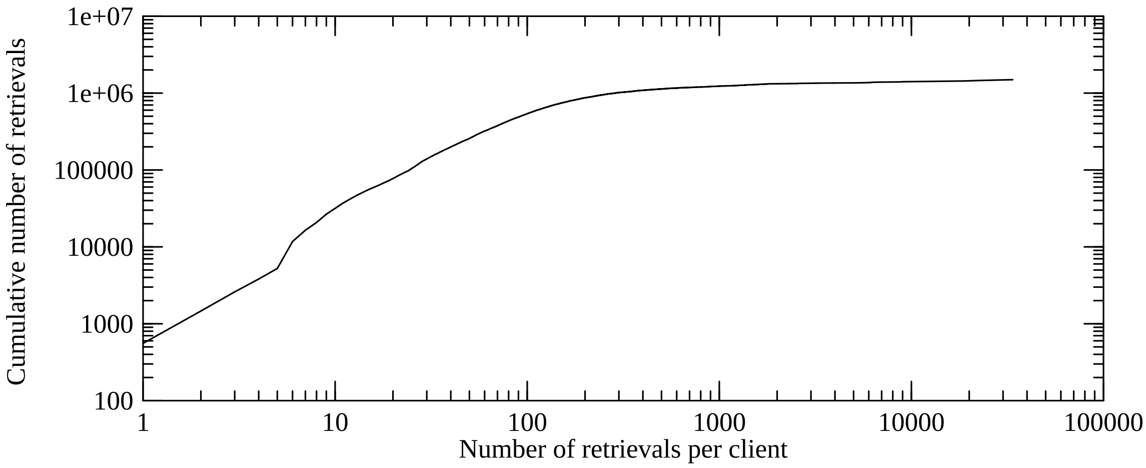


Figure 5-1: Cumulative distribution of client retrieval count

total. Many of these highly active hosts were actually acting as relays, concentrating requests from many different users. For example, the most-active host is a relay for a California-based company.

Others were time-shared machines. Seventeen individual hosts in this category shared one IP subnet, together accounting for 25982 requests, and should probably be treated as a single large timesharing system.

The scatter plots in figures 5-2 and 5-3 show the peak request rates for each of these 90 most-active hosts. The horizontal position of each mark shows the total number of requests made by the host; the vertical position shows the peak request rate. Figure 5-2 shows the 1-second peak rates; figure 5-3 shows the 1-minute and 1-hour peak. The marks enclosed by a diamond reflect the aggregate behavior of the 17-host cluster.

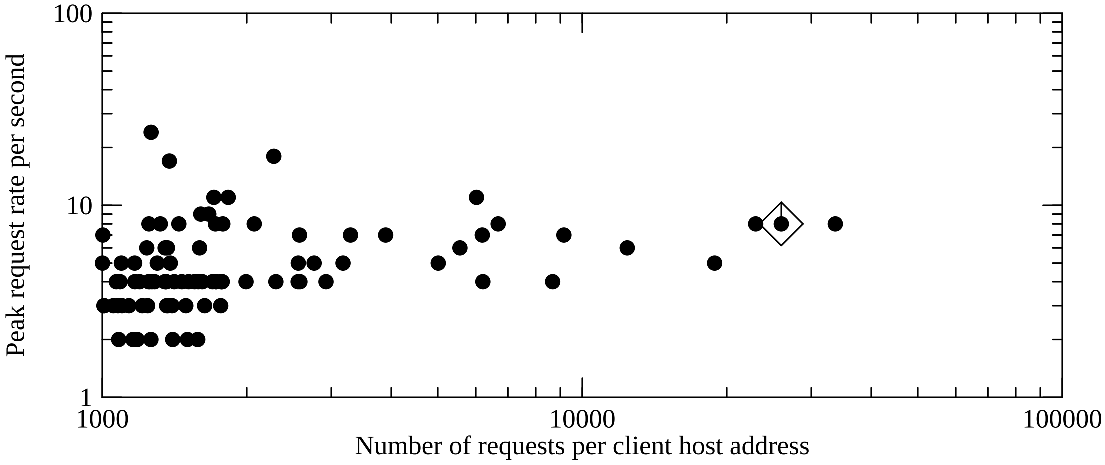


Figure 5-2: Short-term peak request rates for most active hosts

Can we tell if these hosts are single-user systems, timesharing systems, or relays? The similarity in peak request rate between the 17-host timesharing cluster and the most-active host, which is a relay, suggests that this may be difficult. (However, see section 7.3.)

The peak 1-second request rate among these 90 hosts was 24 requests per second. This may seem like a lot of requests for a single user to make, but closer analysis suggests that this was

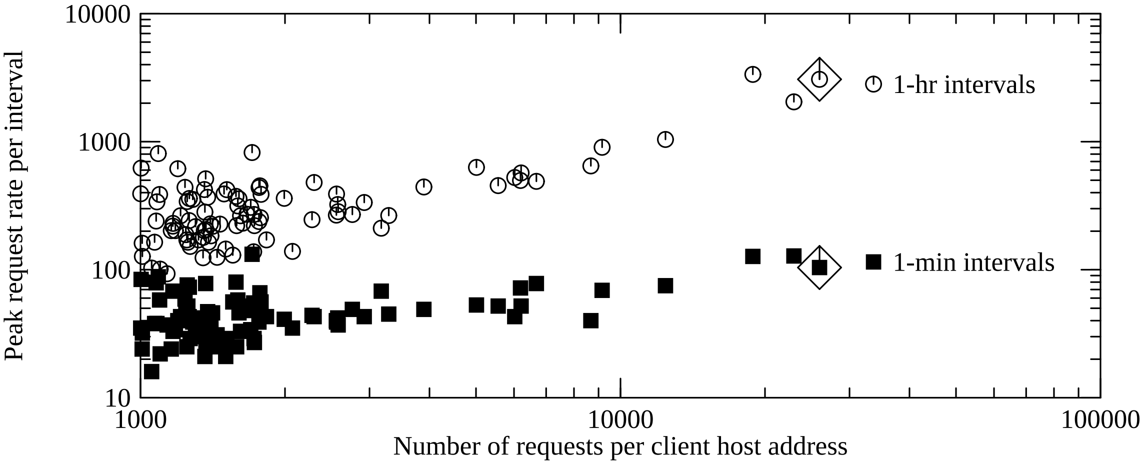


Figure 5-3: Long-term peak request rates for most active hosts

indeed a single-user host. Figure 5-4 shows what happened during this burst. In this figure, the horizontal axis shows time passing; each integral position on the vertical axis corresponds to a unique URL. The lines plotted show the starting and ending times for retrieving a giving URL; fat lines correspond to HTML files.

The first URL retrieved in this timeline shows up as a long thin line; this is a moderately large GIF image showing a map of California. The user evidentally clicked on a county, because the next URL is an HTML file giving the returns for a county. The rest of the URLs, all small GIF files, are the bar-graph images for these returns. (The graphing program assigns URL identifiers in order of request completion, not request starting time, and so the left-hand edge of the resulting curve is ragged.)

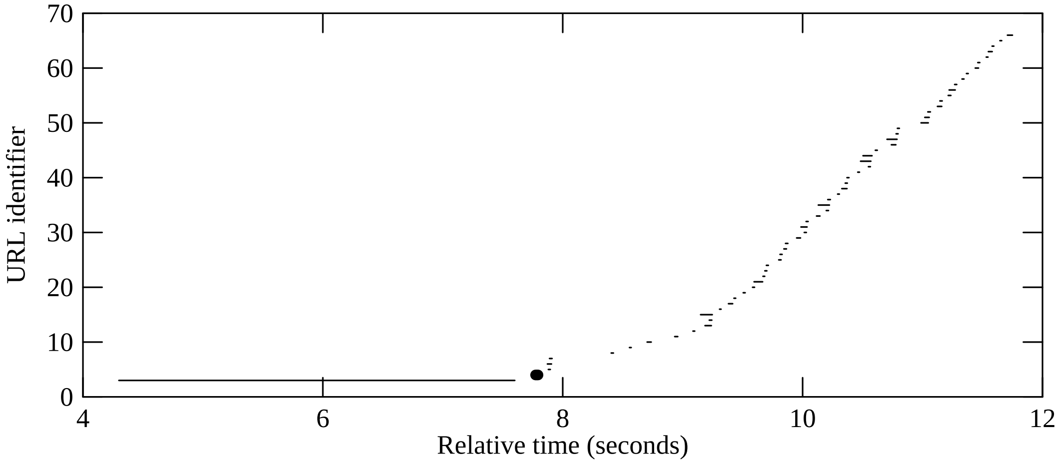


Figure 5-4: Timeline for peak-rate burst from a single client

In the period shown in figure 5-4, the client retrieved each URL exactly once. This client apparently did not often cache inlined images, confounding our efforts to avoid redundant retrievals. Figure 5-5 shows this behavior in more detail. For example, at T = 164 seconds, the user retrieves an HTML file and then a huge number of inlined images, including quite a few that have been retrieved during the previous three minutes. All of these are tiny files, and presumably would have fit in the browser's cache.

We believe that this client was running the *Netscape* browser, since careful examination of figure 5-4 shows that GIF retrievals occurred in bursts of about four simultaneous requests; this is precisely the distinctive scheme that *Netscape* uses to improve perceived latency. The user, who has apparently reached the HTML pages by clicking on county maps, has had to hit the browser's ''Reload'' button to get fresh copies of these pages. In *Netscape*, ''Reload'' causes retrieval of all inlined images in addition to the HTML file. (*Mosaic*, on the other hand, only reloads the HTML file.)

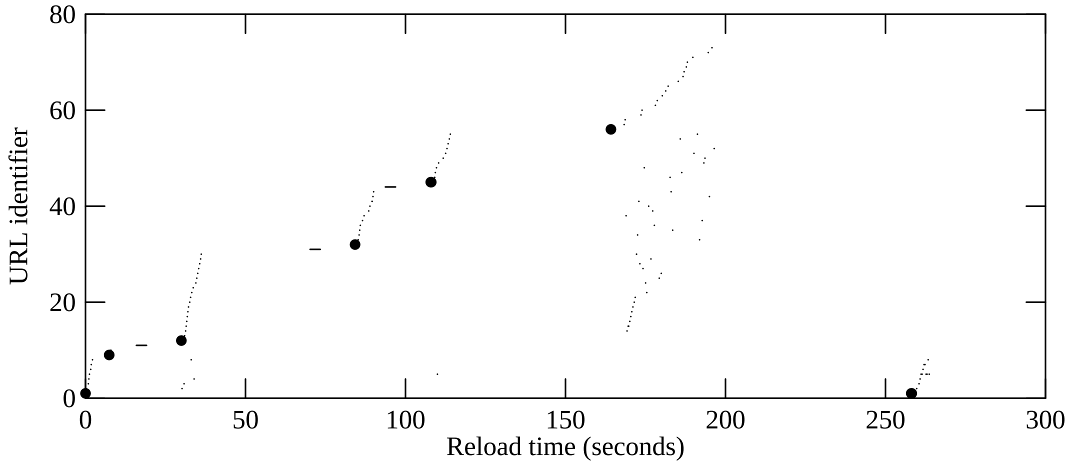


Figure 5-5: Timeline showing lack of effective image caching

This particular client occasionally retrieved the same GIF file several times within the space of a few seconds. Figure 5-6 shows five retrievals of one GIF file, all within less than three seconds. This may be a shortcoming of *Netscape's* simultaneous-connection implementation.

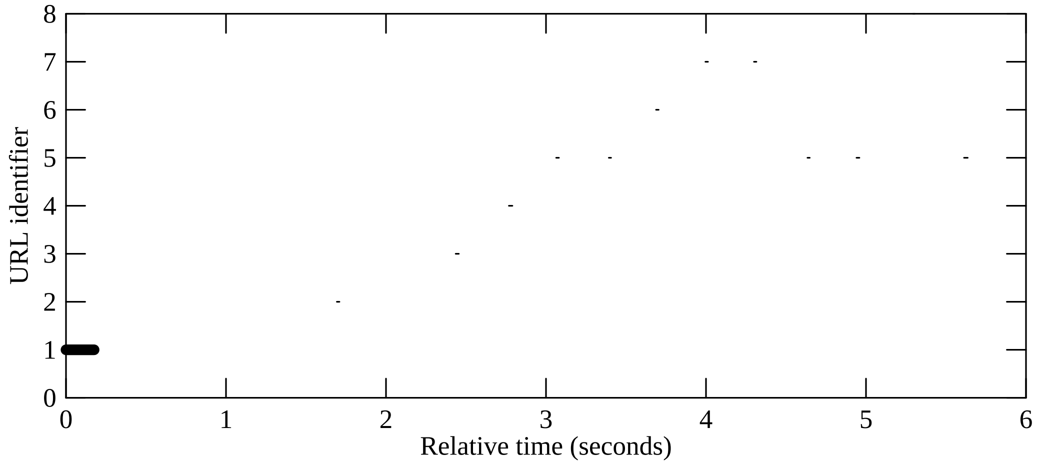


Figure 5-6: Timeline showing short-term redundant requests

Another client displayed even less efficient behavior. Figure 5-7 shows this client doing a short series of retrievals of the page summarizing all statewide races. This page has 45 GIF references, mostly to bar-graph images. There are only 17 unique bar-graph GIF images referenced, however (for example, one party consistently drew about 2% of the voters, and so the 2% bar appears many times). This client blindly issues a request for every GIF reference in the HTML file, without caching the results even for this brief period, and so makes about twice as many retrievals as necessary.

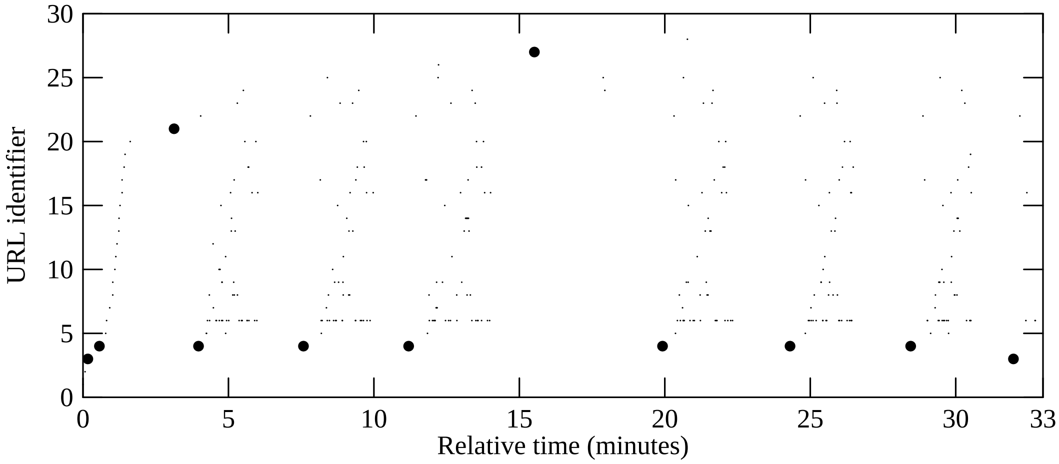


Figure 5-7: Timeline showing lack of any client caching

Suppose the busy clients had perfect caches for the GIF files, static HTML files, and version-numbered dynamic HTML files; how much would this have reduced the server load? (This is not completely infeasible; uncachable files could be marked by the server with a time-to-live of zero.) Figure 5-8 shows, for the 90 most active hosts, how many retrievals would they have been made had they been using a perfect cache. Apparently, they would have made almost an order of magnitude fewer requests. Of the 266176 requests made by these hosts, only 62159 (23%) were strictly necessary.

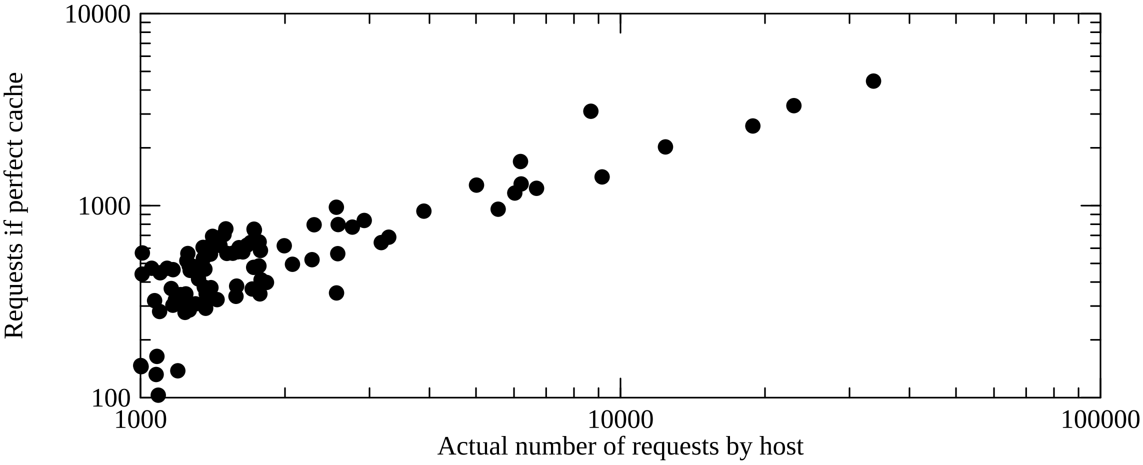


Figure 5-8: Potential effects of perfect caching

Clients that made fewer requests would not have benefited as much from perfect caching, because they did not do as many redundant retrievals. For example, eleven hosts made 155 requests each (the middle of the cumulative distribution in figure 5-1). With perfect caching, they would have made between 56 and 133 requests, with a mean of 103 (66% of the actual number).

Even with perfect caching, busy hosts (both relay and timesharing hosts) make a large number of unique requests. The busiest client host would still have made 4452 requests (instead of 33665). It turns out that many of these requests are for map-based retrievals. Out of the 5944 unique URLs requested by these 90 hosts, 1620 (27%) were via coordinates in county and TV maps. (For figure 5-8, I assumed that these retrievals were not cachable.)

6. Network path statistics

We were interested in the nature and behavior of the paths taken through the Internet between our clients and our servers. We did two kinds of path measurements: post-election measurements of paths to a large subset of the actual clients, and periodic probes during the election to our servers from a few selected sites.

6.1. Post-election path measurements

In an attempt to characterize the actual paths between our servers and their actual clients, I started by extracting 19,070 host addresses from the server logs. This covers the 1,328,862 requests made between 00:00 November 6 and 14:00 November 10.

I then generated scripts that probed the path to each of these hosts. For each host, I ran *traceroute* and two sets of of *ping* trials. *Traceroute* attempts to discover the sequence of routers taken to reach a destination, and also measures the delay to each router and the final destination. However, *traceroute* uses an indirect mechanism and does not always yield a full path; it can also be confused by shifting paths. *Ping* simply sends a series of ICMP Echo [16] packets to the destination, and measures the time until the corresponding ICMP Echo Reply packets come back. I used *ping* to make ten measurements to each destination with each of two packet sizes, 56 bytes and 536 bytes.

Network round-trip time and timeouts (for non-responsive hosts) limit the speed at which these probes can be done. Even though I ran 26 probes in parallel, it took from 18:21 on November 10 to 13:43 on November 16 to collect the results. Clearly, over this period the path characteristics could have changed quite significantly, especially on paths subject to occasional overload. And since these probes were made after the load on the election servers had declined significantly from its peak, the network state during the probing could have been quite different from its state during the peak election load. However, we did not want to complicate things by doing the path-probing during the period of peak load.

Another consequence of doing the probes several days after the peak load is that many of the IP addresses failed to respond to any probes. I only received *ping* responses from 52% of the hosts, and complete *traceroute* paths from 44%. (43% gave us both kinds of information). Non-responsive host addresses might be behind firewalls, or might have been connected by dialups, or might have been dynamically assigned for brief durations.

6.1.1. *Traceroute* results

I obtained full *traceroute* paths from 8329 hosts, representing 709,132 retrievals (53% of the total). Figure 6-1 shows the distribution of path lengths. (The horizontal axis terminates at 30 hops, because I limited *traceroute*'s probes to that depth to avoid lengthy delays.) Figure 6-2 shows the same distribution, weighted by the number of retrievals. The mean path length, averaged over the 8329 hosts, was 12.85 hops. The mean path length weighted by the number of retrievals done by the responding host was 10.76 hops. In other words, clients that were further away tended to make fewer requests.

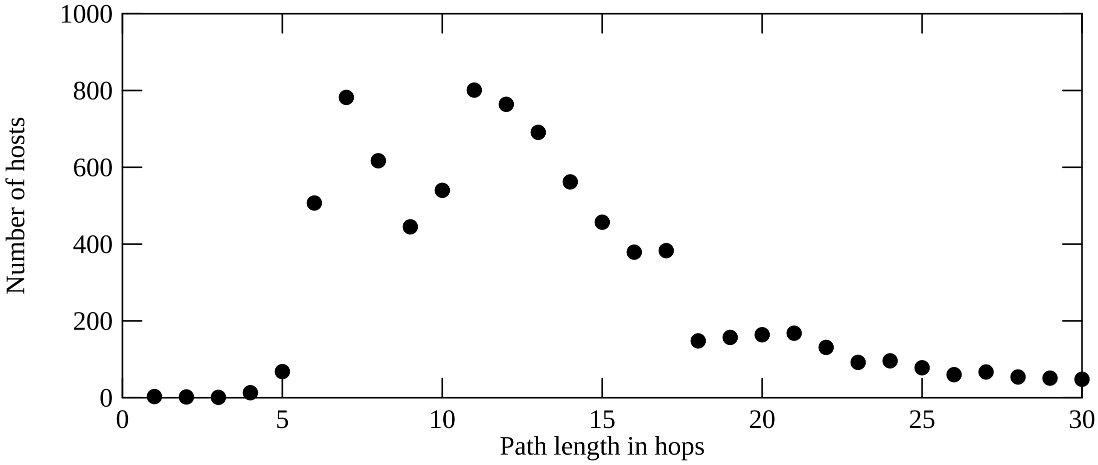


Figure 6-1: Distribution of path lengths to clients

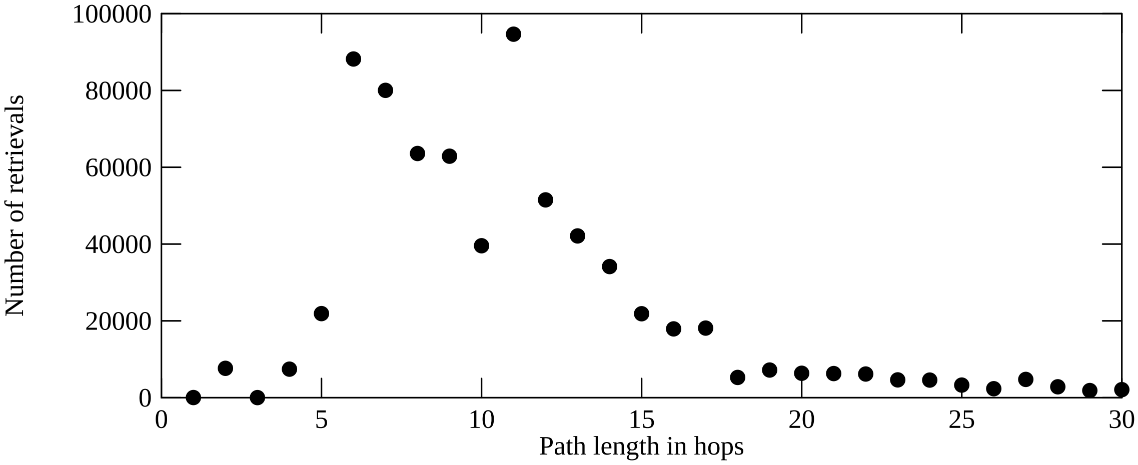


Figure 6-2: Weighted distribution of path lengths to clients

6.1.2. *Ping* results

Of the hosts that responded at least once to an ICMP Echo packet, the average host replied to about 9.5 of the 10 Echos sent in each trial. For the 9840 hosts responding to 56-byte pings, the mean round-trip time was 207 msec; weighted by the number of retrievals the mean was 146 msec. That is, clients with lower delays tended to make more requests. For the 9755 hosts responding to 536-byte pings, the mean delay was 322 msec, and the weighted mean delay was 229 msec.

Figure 6-3 shows the distribution of round-trip times for both packet sizes. 56-byte data are marked with filled circles; 536-byte data are marked with open squares. Note that the distributions are trimodal; the 56-byte values peak near 10 msec, 80 msec, and 600 msec. The 536-byte values peak near 25 msec, 100 msec, and 600 msec.

9726 hosts responded to both sizes of ping packet. With delay measurements made using two different packet sizes, we can compute the apparent bandwidth of the path taken by the probes. The bandwidth estimate is simply the difference in the packet sizes divided by the difference in minimum delays, then doubled (because the data takes a round trip; this assumes a symmetrical path). Figure 6-4 shows the distribution. I tried using mean delays instead of minima, and got a similar curve.

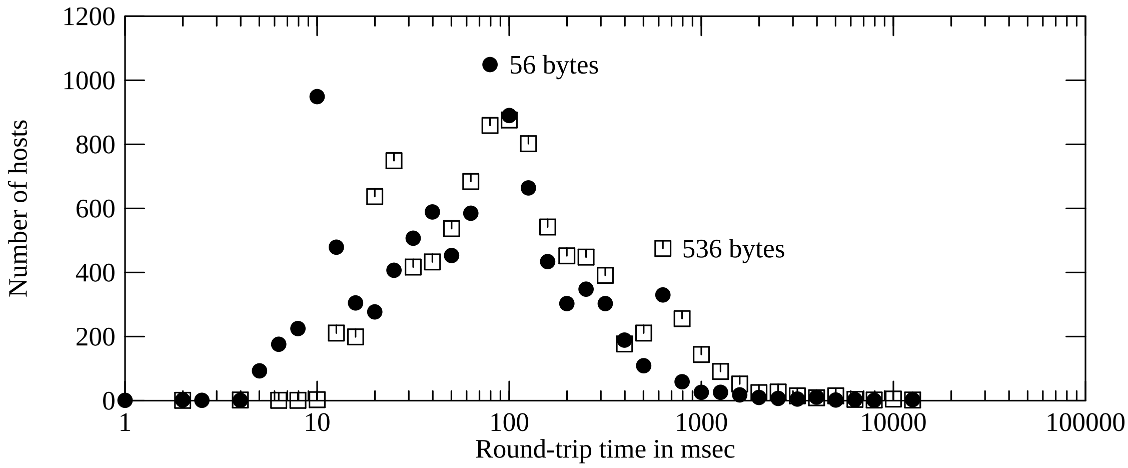


Figure 6-3: Distribution of round-trip times

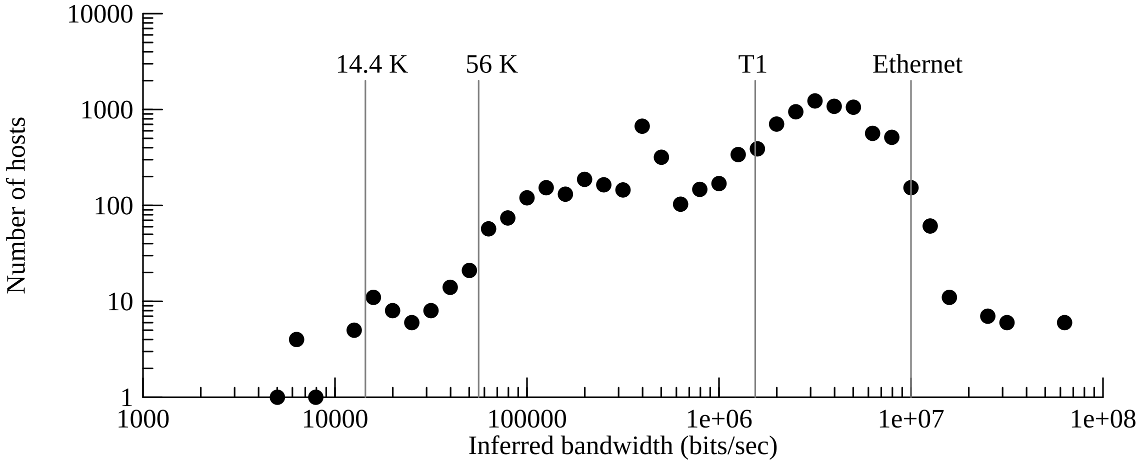


Figure 6-4: Distribution of inferred bandwidths

Figure 6-4 shows several distinct features. The bandwidth estimates above 10 Mbit/sec are clearly bogus, since they are certainly the result of noise in the measurements; all paths included at least one hop on a 10 Mbit/sec Ethernet. None of the delays were below 6 Kbit/sec, suggesting that relatively few of the hosts that responded to probes were on low-speed dialups. The distribution shows several peaks, one at approximately 15 Kbit/sec, a broad peak around 200 Kbit/sec, a sharp peak at 400 Kbit/sec, and a broad one between 3 Mbit/sec and 5 Mbit/sec. This distribution suggests that a large portion of the client hosts are well-connected to the Internet, via fractional T1, full-speed T1, or faster links. However, since only about half of the clients responded to post-election *pings*, low-speed (dialup) hosts probably represent a much larger fraction than this distribution indicates.

Finally, I looked at the correlation between path length and estimated bandwidth. Figure 6-5 shows the distribution of results; the area of each circle is roughly proportional to the number of hosts with a given hop count and delay. There seems to be a slight negative correlation for high bandwidths (hundreds of Kbits/sec) and path lengths between 6 and 9 hops. Otherwise, bandwidth seems to be independent of path length. This is probably many ‘‘long’’ paths spend most of their hops in well-connected regional trunk networks.

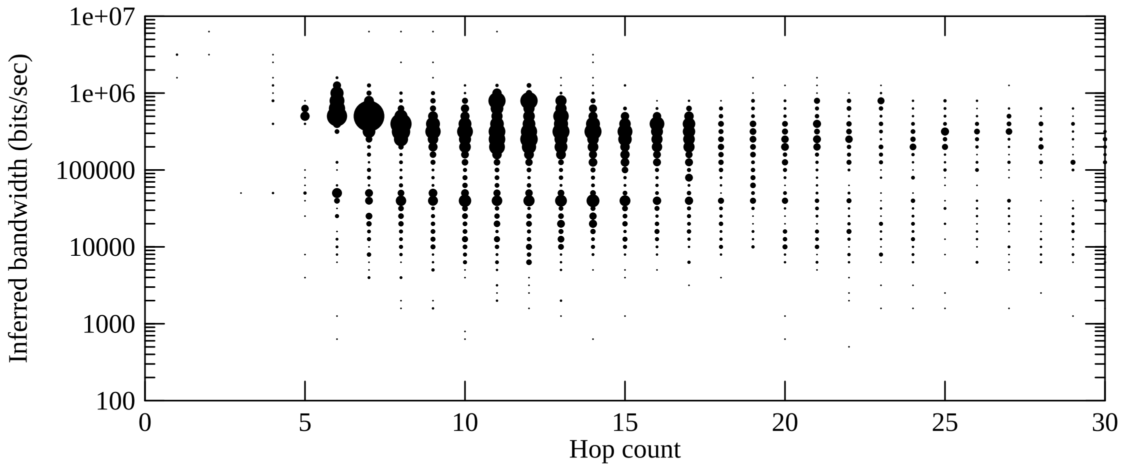


Figure 6-5: Correlation between hop count and estimated bandwidth

6.2. Periodic probes during the election

For a period of about two days (November 8 and 9), we ran periodic probes to our servers from three sites on the Internet: Software Tool & Die in Massachusetts (`world.std.com`), Marquette University in Wisconsin (`marque.mscs.mu.edu`), and the University of Pennsylvania (`upenn.edu`). Every 15 minutes, the probe scripts ran *ping* to our server to measure the round-trip time for ICMP Echos of several different sizes, and ran a simple test program to measure the retrieval time for several pages from the server. One page was a 3KB HTML file; the other was a 7KB GIF file.

The probe scripts also used *traceroute* to discover the path taken between the probe sites and our server. The path from `upenn.edu` consistently took 15 hops. The path from `marque.mscs.mu.edu` varied continually, between 16 and 18 hops. The path from `world.std.com` started at 6 hops, but shifted to a 9-hop path late on the evening of November 9.

The top graph in figure 6-6 shows how the sampled round-trip time values varied over time (these measurements are for minimal-sized ICMP Echos). Note that the horizontal axis shows Eastern Standard Time, not Pacific Time. Filled circles show data from `marque.mscs.mu.edu`; plusses show data from `upenn.edu`; open squares show data from `world.std.com`. The RTTs increase slightly during the periods of heavy load on our server. The bottom graph in figure 6-6 shows the cumulative distribution of round-trip time samples.

The top graph in figure 6-7 shows how retrieval times for the HTML file varied over time. The bottom graph shows the cumulative distribution. The time axis in each of these graphs is on a log scale, because in a few cases, retrievals took many minutes. This probably reflects episodes of heavy packet loss somewhere in the network, or perhaps a temporary loss of network connectivity. Retrieval times in the range of one to ten seconds were distressingly common, suggesting that users might not have always been pleased with response time. However, most of the retrievals did take less than one second.

Because we had *ping* measurements for several different packet sizes, I was able to infer the network bandwidth between these client hosts and our servers. Figure 6-8 shows the results.

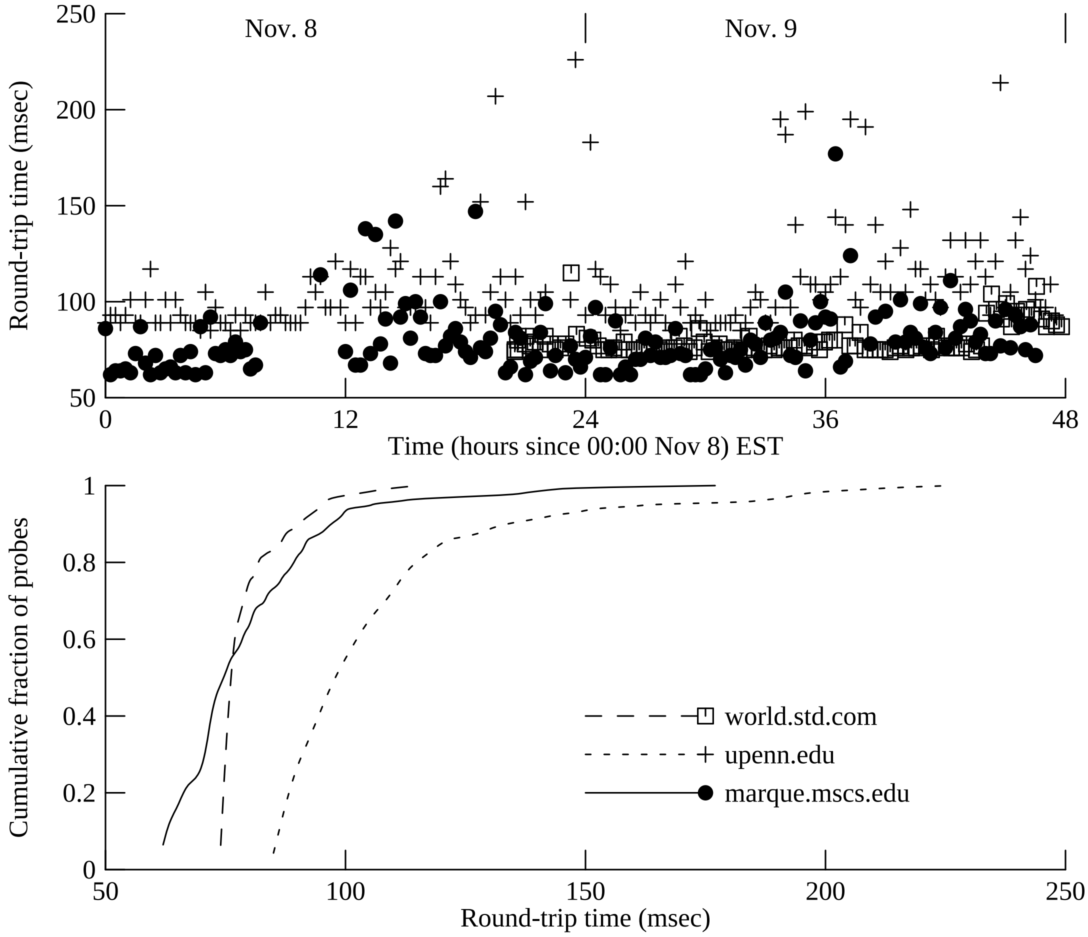


Figure 6-6: Sampled round-trip times from periodic probes

Note that this method may not adequately reflect network congestion, since it measured RTTs for single packets. A packet-pair approach [3] would have been more appropriate.

7. Interarrival patterns

Several recent studies [8, 10, 14] have suggested that packet interarrival times tend to fit distributions other than Poisson. Does the data from our traces confirm this?

Using the server logs (with 1-msec timestamp resolution) and the *tcpdump* traces (with 1-usec resolution), I was able to obtain interarrival distributions for several categories of events. These distributions support the observations by others that real networks with real users cannot be modelled by a Poisson process.

7.1. Packet arrivals

Because of the immense volume of the *tcpdump* traces, I concentrated on the data for just the busiest date, November 9[2]. I looked at the arrivals of several different classes of packets, for the

[2]A sequence representing 3.5% of the traced packets was left out, due to disk space limits on the analysis system.

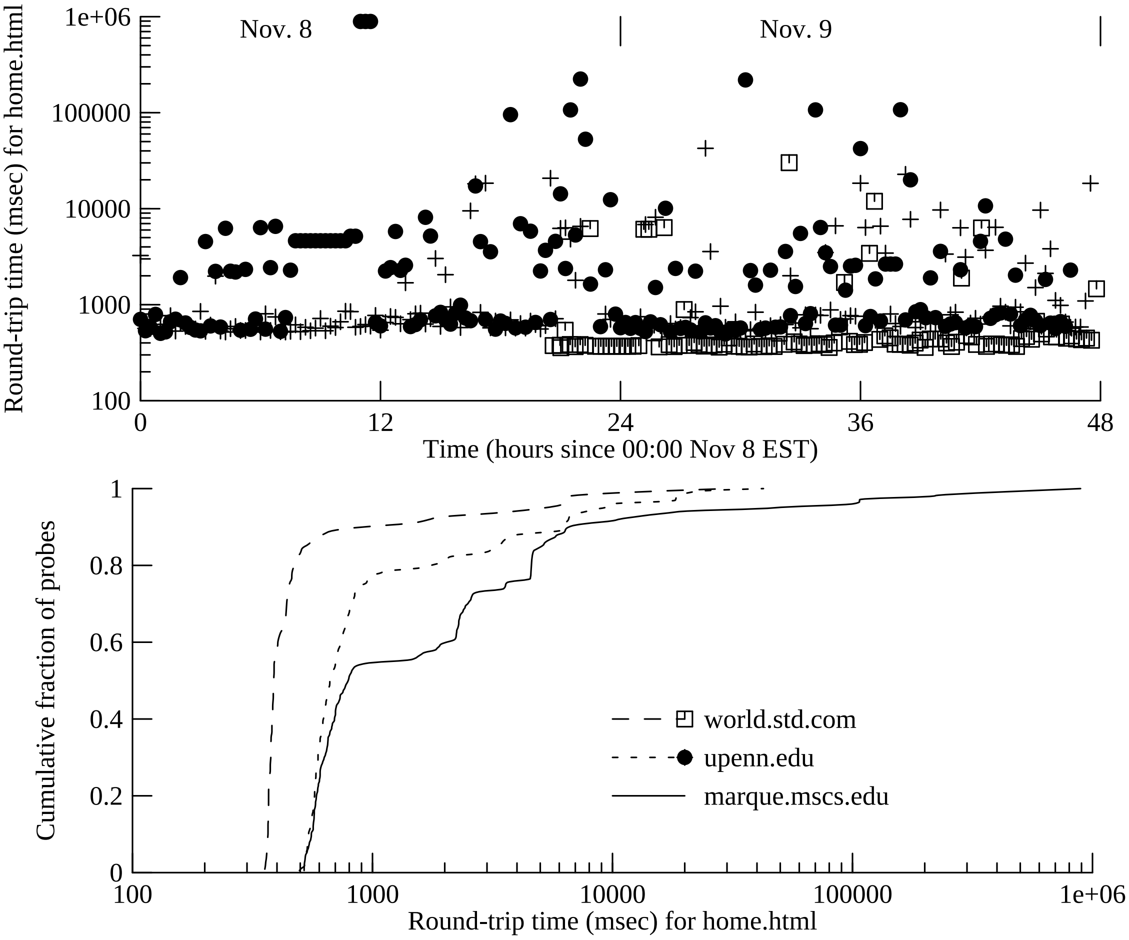


Figure 6-7: Sampled HTML retrieval times from periodic probes

three server hosts. These classes include: all incoming HTTP packets, all outgoing HTTP packets, all incoming HTTP packets with the TCP SYN bit set (representing a connection request), and all outgoing HTTP packets with the SYN bit set (representing acceptance of a connection request). (Although the traces only show that the incoming packets appeared on the LAN, prior experience suggests that nearly all of these packets were in fact received by the server kernels, although they may have then been discarded due to resource limits.)

Figure 7-1 shows the cumulative distributions of interarrival times for these four classes. The median of each distribution is marked on the curve; the mean interarrival time is shown in the key. The thick gray line shows a Poisson distribution with the same mean interarrival time as the ‘‘all incoming HTTP packets’’ curve (13 msec); it has a much different shape than the actual data.

Figure 7-2 shows the same distributions, but for a particularly busy 1-minute period. Again, the Poisson distribution with the same mean rate as the ‘‘all arriving HTTP packets’’ curve is shown by a thick gray line. Observe that these curves generally have the same shape as those for the full 24-hour period, even though the mean rates are higher.

Figure 7-3 shows the same distributions as in figure 7-1, but as a histogram. The solid line on this figure shows the Poisson distribution with the same mean arrival rate as the ‘‘all arriving HTTP packets’’ curve. Again, note that the actual data does not correspond to the Poisson distribution: it has a much broader tail, and shows several sharp peaks for small arrival rates.

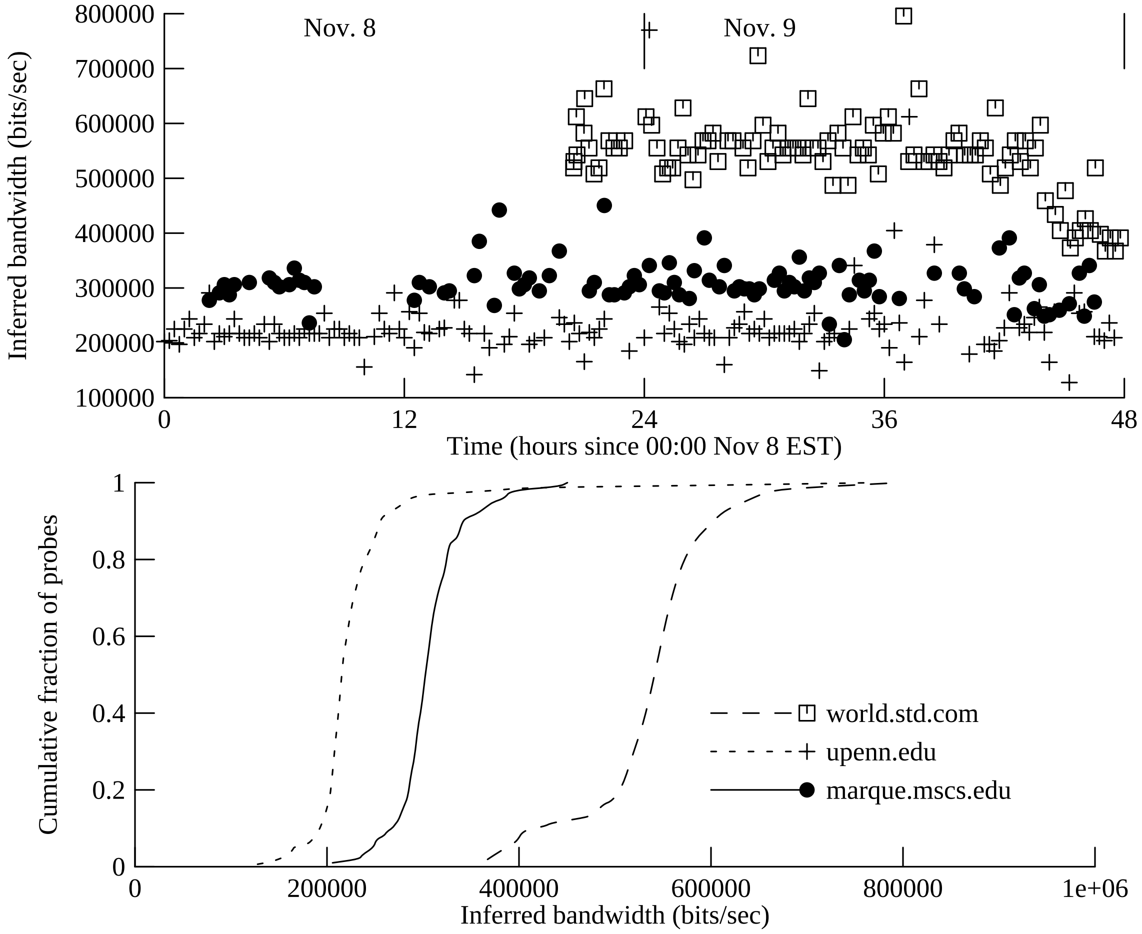


Figure 6-8: Network bandwidths inferred from periodic probes

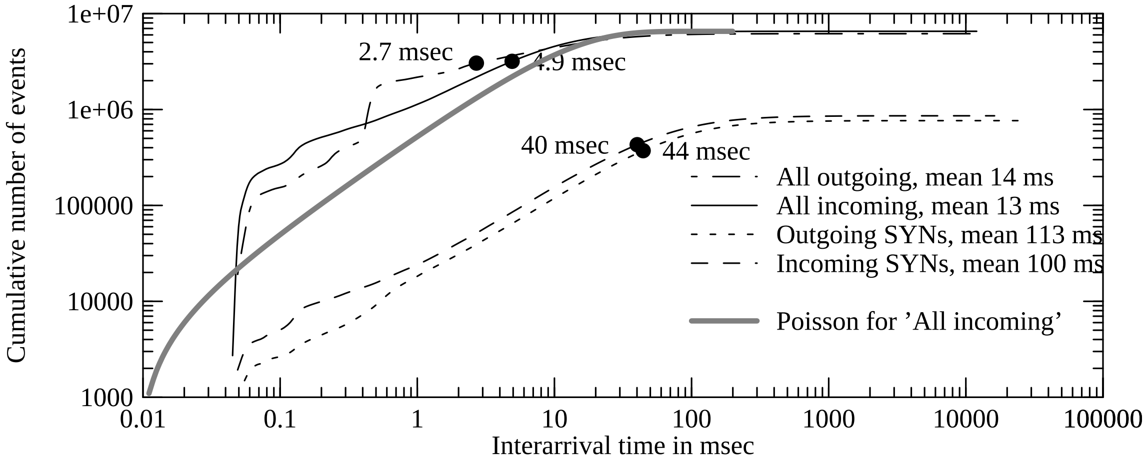


Figure 7-1: Cumulative distributions of interarrival times, Nov. 9

Figure 7-4 shows the same distributions, but for the busy 1-minute period. In this graph, we can see that the Poisson curve is a fairly good fit for the incoming packet arrivals, although it tends to underpredict the number of very short and very long interarrival times.

The real arrival rates for incoming packets differ obviously from the Poisson curve with several sharp peaks for small interarrival times. These peaks may correspond to the peaks in the

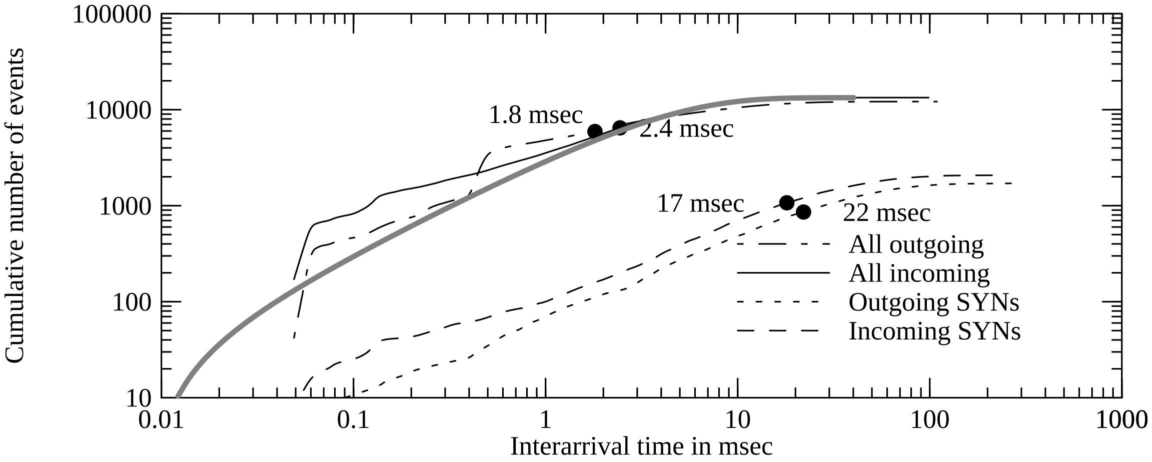


Figure 7-2: Cumulative distributions of interarrival times, Nov. 9 09:48

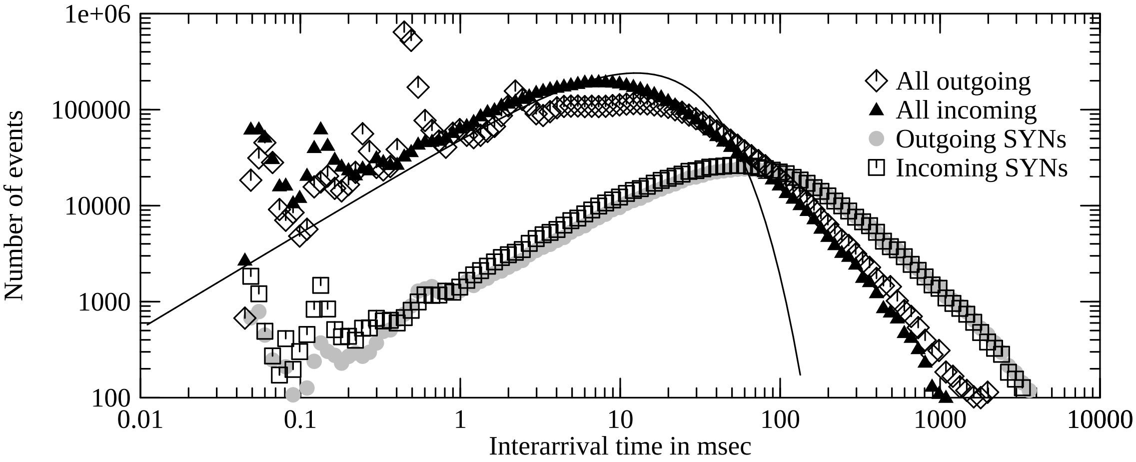


Figure 7-3: Distributions of packet interarrival times, Nov. 9 (same data as figure 7-1)

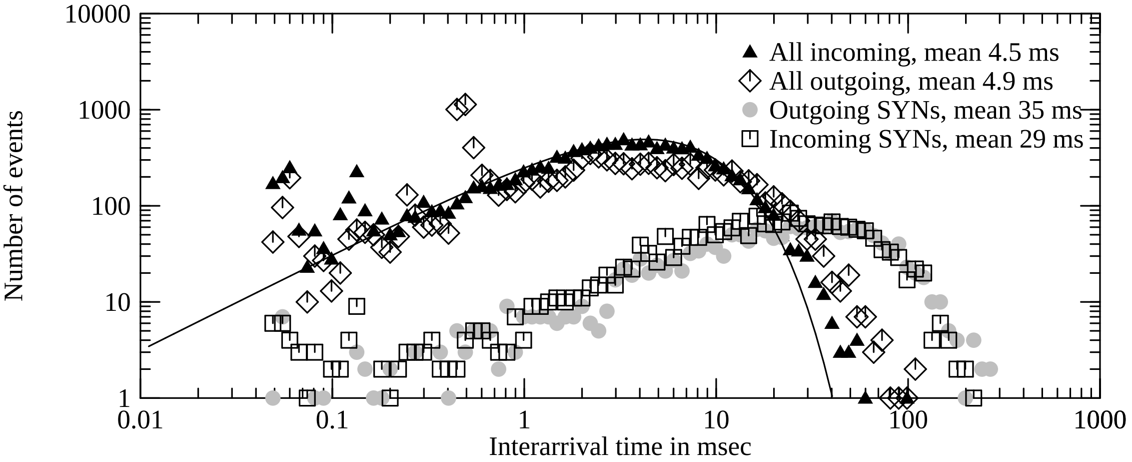


Figure 7-4: Distributions of packet interarrival times, Nov. 9 09:48

packet-size distributions, shown in figures 7-5 and 7-6. The distribution of transmitted packet sizes shows sharp peaks at 60, 300, and 588 bytes, corresponding to packet durations of about 70, 260, and 490 usec. The ‘‘all outgoing’’ distribution in figures 7-3 and 7-4 show peaks at these interarrival times.

The ‘‘all incoming’’ and ‘‘all outgoing’’ distributions also sharply peak at about 120 usec. This may be a reflection of our routing topology: since we had two different routes to the Internet, the servers often chose the wrong outbound router. This router would issue an ICMP Redirect to the server, and also retransmit the packet across the Ethernet to the correct router. Thus, numerous outgoing packets appeared twice, but these were mostly short (60-byte) packets since the subsequent longer packets followed the Redirected route.

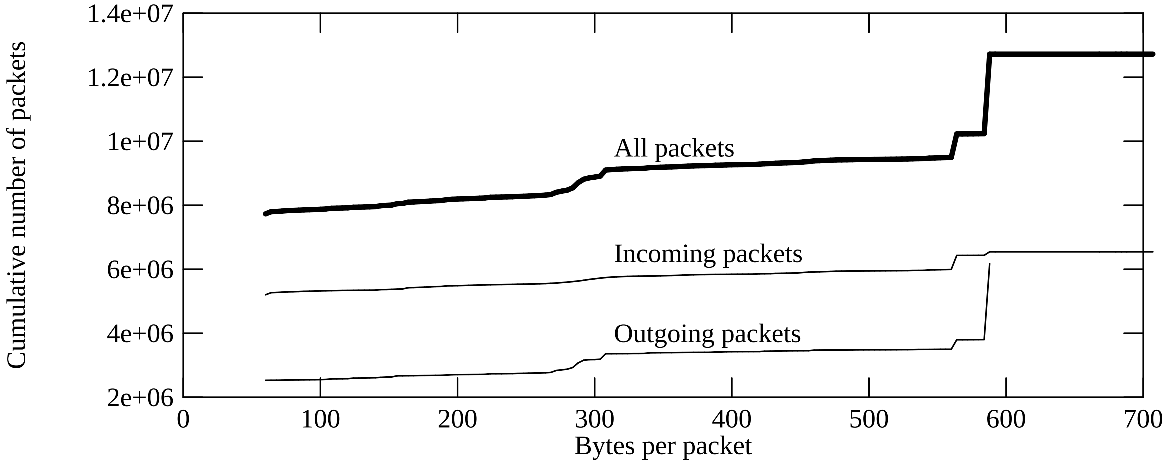
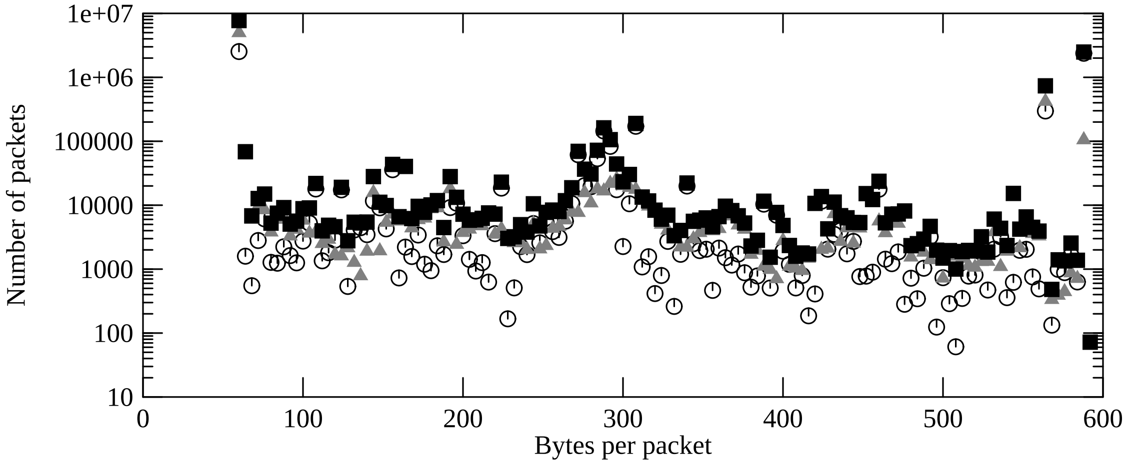


Figure 7-5: Cumulative distributions of packet sizes, Nov. 9



Circles: outgoing packets; Triangles: incoming packets;
Squares: all Election-server packets

Figure 7-6: Distributions of packet sizes, Nov. 9

7.2. Request arrivals

Figure 7-7 shows the interarrival times for new HTTP requests, as seen by the HTTP server daemons themselves. The open circles show the arrivals seen by all server hosts taken together (the host clocks were synchronized using NTP, probably to better than 1 msec [12]). The dotted line shows the Poisson distribution with the same mean arrival rate. The solid triangles show the arrivals seen by just one of the servers; the solid line shows the corresponding Poisson distribution.

The real distributions tend to follow the Poisson curves for interarrival times below the mean, but have much larger tails. The real data also shows two large peaks, at 1 msec and at about 7 msec. The 1-msec peak is a measurement artifact of the 1-msec timestamp resolution, and corresponds to the number of events with all interarrival times <= 1 msec. The 7-msec peak probably reflects the CPU-time cost to dispatch a new process for each request; note that the single-server distribution shows almost no interarrival times below about 6 msec. These two distributions imply that at short time-scales, requests arriving at the server *hosts* do follow a Poisson distribution, but queueing delays in the servers cause the server *processes* to see a non-Poisson distribution.

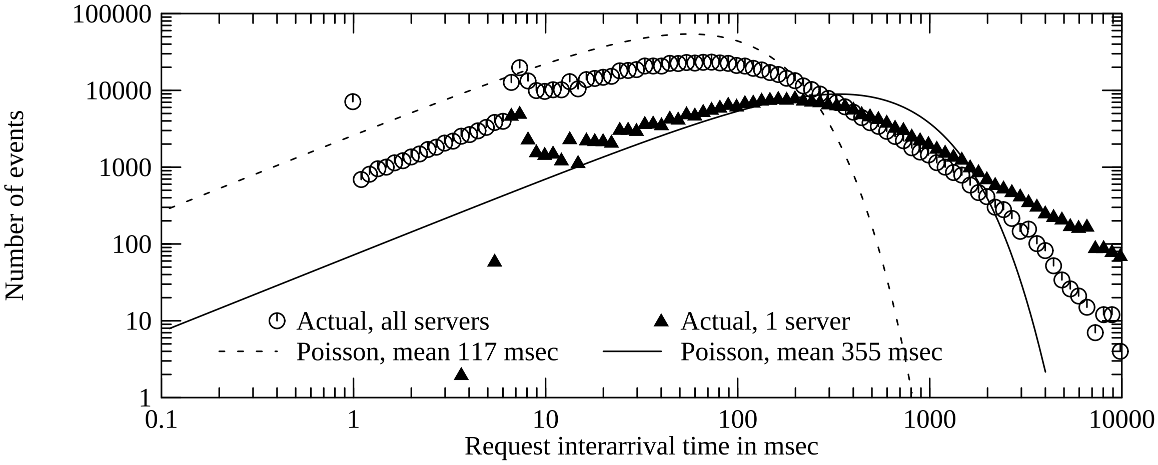


Figure 7-7: Distributions of request interarrival times, Nov. 9

7.3. Per-client request arrivals

In addition to the overall arrival pattern, I looked at the interarrival time distribution for requests from several individual sources mentioned in section 5: the busiest proxy, the 17-host timesharing cluster, and the single-user host with the highest peak request rate. These distributions are shown in figure 7-8, along with Poisson distributions with similar mean interarrival times.

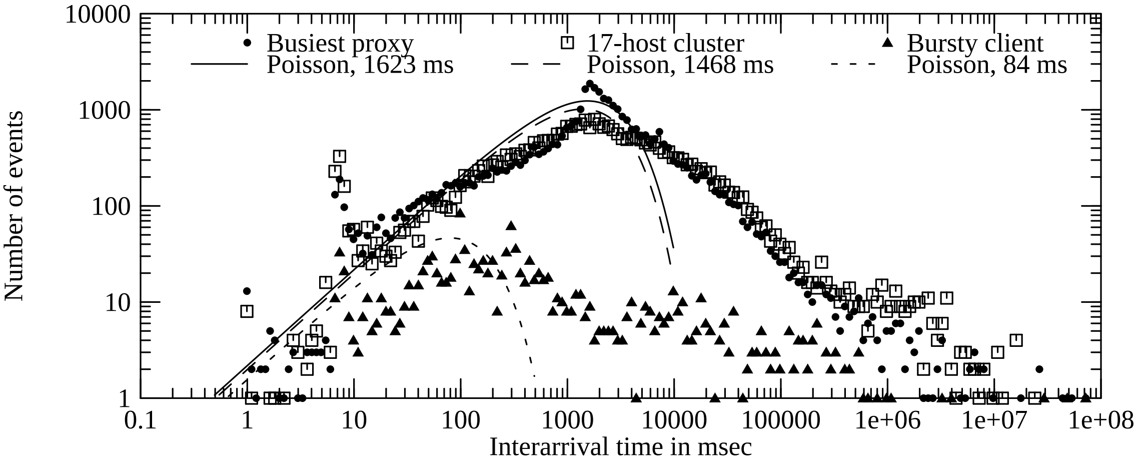


Figure 7-8: Distributions of request interarrival times, selected clients

Figure 7-9 shows the same data as in figure 7-8, as cumulative distributions.

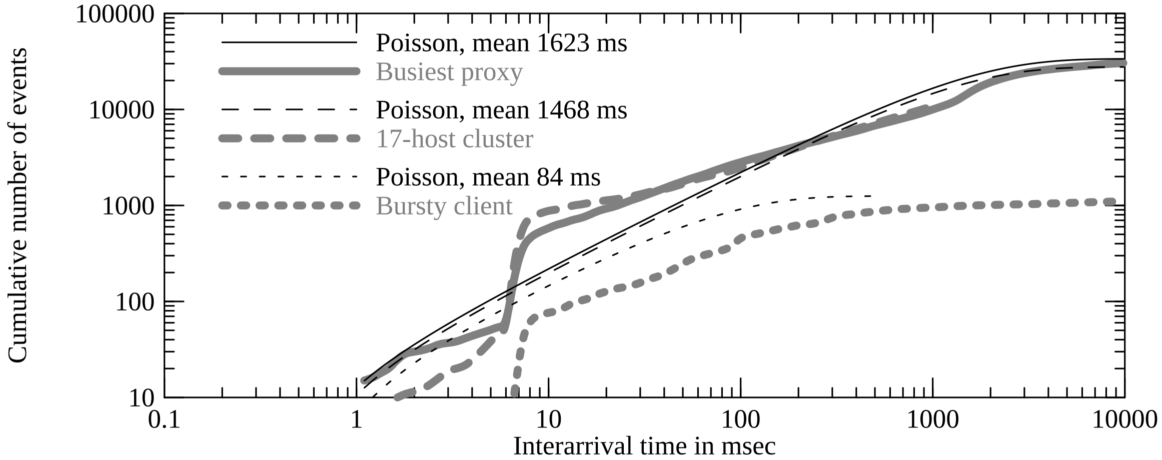


Figure 7-9: Cumulative distributions of request interarrivals, selected clients

Once again, the actual distributions deviate significantly from the Poisson distributions for interarrival times much above the mean, but match fairly closely for smaller values. Also note that the single-user system has a much lower mean than either of the multi-user distributions, and the busy proxy has a much sharper peak than the timesharing cluster. A straightforward proxy server must funnel all requests through a single synchronization bottleneck, causing queueing delays not seen in a timesharing system. This ‘‘signature’’ in the interarrival time distribution may allow one to distinguish proxies from other kinds of clients.

7.4. Summary of arrival data

While we have not yet done extensive analysis at numerous time scales, it does appear that the packet interarrival times we measured do diverge from a pure Poisson process, especially for longer measurement periods, and they show an excess of very short and very long interarrival times. Likewise, HTTP request interarrival rates deviate from Poisson, especially when viewed by the server process itself.

Recent work by others has suggested that network arrival patterns are self-similar rather than Poisson [10], and that the self-similarity arises from the behavior of ‘‘on/off’’ sources [17]. Crovella and Bestavros have shown this specifically for other HTTP traces [5]. One characteristic of such sources is a long tail in the interarrival time distribution, such as is seen in figures 7-7 through 7-9.

8. TCP behavior patterns

Because we had extensive *tcpdump* traces, this provides an opportunity to look at the details of client and server TCP behavior under conditions of heavy server load.

This section includes statistical information about the number of packets per connection, inferences about the cost of searches in the servers’ PCB tables, and the frequency and arrival times of TCP retransmissions.

8.1. Packets per connection

I started by counting the number of packets per connection, both received and transmitted (from the point of view of the server). The distribution and cumulative distribution are shown in figure 8-1. The mean number of packets per connection was 17; 8.75 from the client to the server, and 8.26 from the server to the client. For the median connection, the client sent between 6 and 7 packets and the server sent between 5 and 6 packets, with a total of between 12 and 13 packets exchanged. The means are larger than the medians because a few connections transferred hundreds or thousands of packets.

These counts include retransmissions; this increases the number of packets per connection (see section 8.3). The counts also include ‘‘connections’’ that failed to complete; these connections often (but not always) involve just a few packets, which could tend to push the medians down. A successful HTTP transaction seems to involve 9 or 10 TCP packets (although fewer would be possible with more effective piggy-backing of ACKs).

About 6% of the ‘‘connections’’ traced seem to have exchanged too few packets to have been successful. Some or most of these apparently too-short connections might be artifacts of the trace-analysis process, which analyzes each sequence of 1 million packets independently and so may truncate connections that take place near one end of such a sequence. A few others may be artifacts of the trace-collection process, which did occasionally lose packets (see figure 3-1).

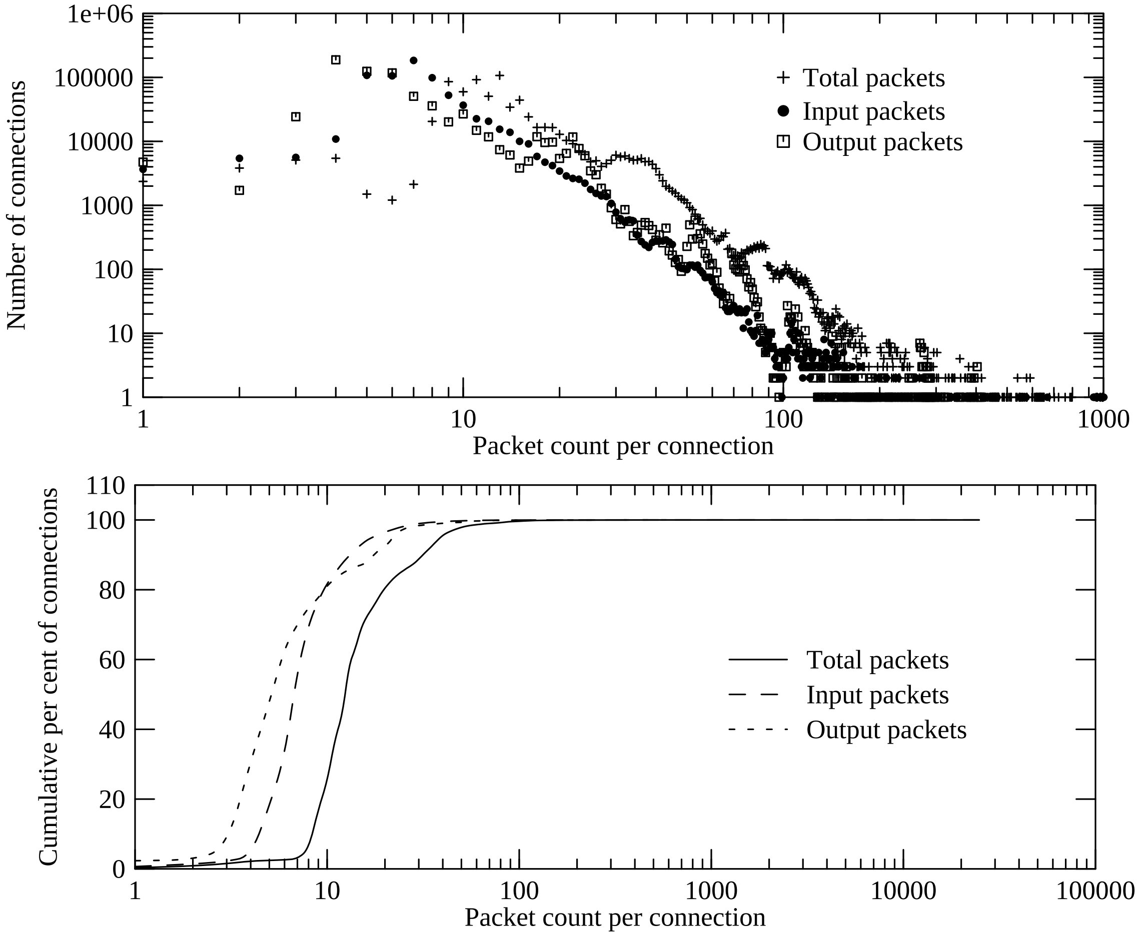


Figure 8-1: Distribution of packets per HTTP connection, Nov. 9

The relatively small number of packets per connection has several implications:

- Any kernel modifications that speed up the handling of packets for an existing connection ought not to significantly increase the cost of creating and deleting connections.
- Most of these TCP connections will be operating in the ‘‘slow-start’’ regime [7], in which the sender uses an artificially small window, and so will not be able to obtain full network bandwidth, especially over high-delay paths.

8.2. PCB table search costs

The kernels running on the servers used a linear search of the PCB table, which is known to impose high CPU costs when the table is large (see 4.1); how large is this cost?

Each received TCP packet causes a search for a PCB table entry, and so this search cost is on the critical path for replies to incoming packets. In particular, the speed with which a server responds to a client’s SYN packet depends only on the cost of handling packet headers and doing PCB table operations, not on the costs of handling packet data or synchronizing with application processes. Therefore, I analyzed the *tcpdump* traces to measure, for each arriving SYN, the time it took one of the server systems to respond. The distribution is shown in figure 8-2.

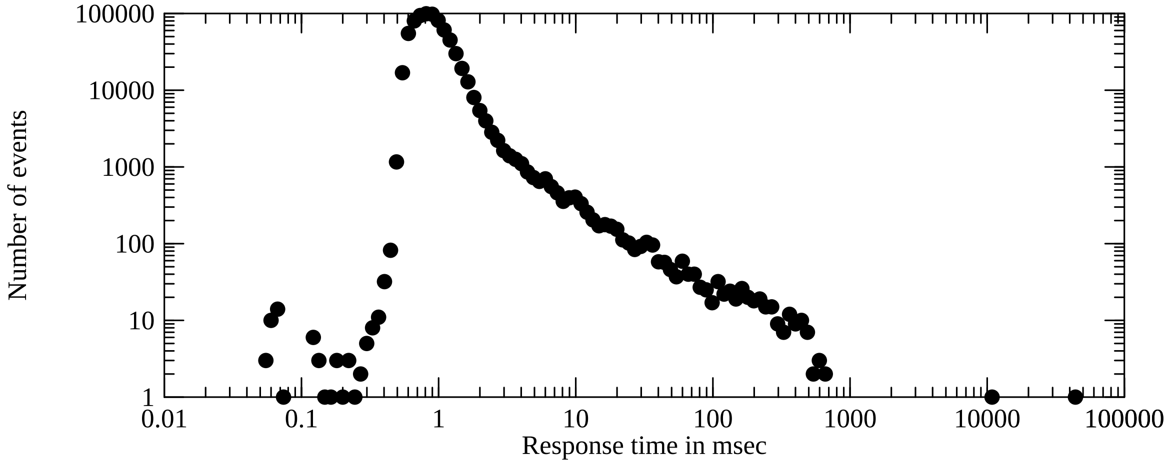
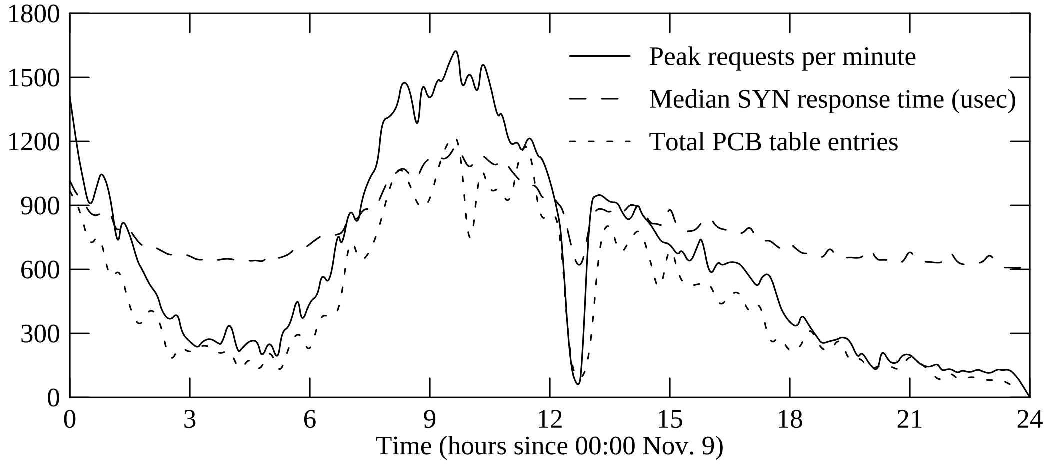


Figure 8-2: Distribution of servers’ response times to SYNs, Nov. 9

The distribution has a sharp peak, at about 800 microseconds. There does not appear to have been many responses that could have been delayed by lengthy table searches; 72% of the responses took under 1 msec, and 97% took under 2 msec; less than 1% took longer than 5 msec. This puts an upper bound of 28% on the number of searches delayed more than 1 msec; since much of the delay could be cause by queueing in the network or in the server, the actual fraction could be much smaller.

But consider the implications if the linear-list PCB lookup mechanism actually did impose an extra 1 msec of CPU time per arriving TCP packet. Since our server received about 7 packets for a median connection (see figure 8-1), this would imply 7 msec of ‘‘wasted’’ CPU time per connection. Because the baseline per-packet processing cost seems to be about 800 usec, this means that the CPU could handle at most (1000/(7 * (1 + 0.8))) or about 79 connections per second. Without the excess PCB lookup costs, the limit would be (1000/(7 * 0.8)), or about 179 connections/second. Experiments with simple test programs suggest that a single-CPU system does indeed bottleneck at about 80 TCP connections per second, even without doing any HTTP processing.

The distribution in figure 8-2 shows that for most of the connections in the trace, the response time was actually quite good. However, the connections that did suffer from poor per-packet response time were probably those in progress during periods of heavy load, when the PCB table size, and thus the excess PCB lookup cost, is greatest. This is precisely the wrong time to impose extra costs on connections, since the system's resources are already stretched. Figure 8-3 shows this correlation between load, PCB table size, and response time.



Request rate: peak 1-minute rate over 10-minute intervals
PCB table size: sampled at 15-minute intervals
Response time: median response time over 10-minute intervals

Figure 8-3: Correlation between request rate, PCB table size, and response time

8.3. TCP retransmissions

TCP senders retransmit packets when they do not receive an acknowledgement within a timeout interval. The timeout value is a function of the sender's estimated round-trip time (RTT), following algorithms that have been developed over years of experience and analysis [7, 9].

One can learn several things about the network and its packet sources by observing TCP packet retransmissions in *tcpdump* traces, and analyzing the retransmission counts and interarrival times.

8.3.1. SYN retransmissions

It is fairly easy to look for retransmissions of TCP SYN packets from clients, and of the servers' SYN|ACK responses. Figure 8-4 shows the distribution of the number of SYNs and SYN|ACKs per connection, including ''connections'' that were never successfully established. The filled circles plot the number of SYNs seen before the first SYN|ACK response; the open diamonds plot the number of SYN|ACK responses seen.

The *y*-axis is a log scale; note that less than 7% of the connections involved a retransmitted SYN, and less than 4% involved a retransmitted SYN|ACK. Almost none of the connections required more than five retransmissions.

Figure 8-5 shows the distribution of SYN interarrival times (including the cumulative distribution), for those connections where at least two SYNs were seen. (By ‘‘interarrival time’’ here I mean the interval between the original transmission, and the first retransmission.) The distribution peaks at about 3 seconds, which is consistent with the recommended initial estimate [4]. Since the SYN is the first packet of the connection, the sender should use a conservative initial estimate, to avoid retransmitting too soon.

About 4% of the clients’ retransmitted SYNs, however, arrived within 1 second of the previous SYN. In particular, the distribution shows a moderate peak at about 150 msec. Additional log analysis showed that almost all of these retransmissions came from clients on a single network. These clients appeared to be using an excessively small value for the initial retransmission timeout. They were, in fact, receiving the server’s first SYN|ACK, but not before retransmitting their SYNs.

These clients have other idiosyncrasies; they set the TCP ‘‘Push’’ flag on every packet (including SYN, FIN, and Reset packets), and they respond to the server’s FIN with Reset if they have already initiated a subsequent connection. In short, this appears to be a small and immature TCP implementation, perhaps on a personal computer, and should not be taken as typical.

The few retransmission events with very short interarrival times might be caused by these excessively short timeouts, coupled with the usual variation in delays through the Internet (which often exceeds 150 msec).

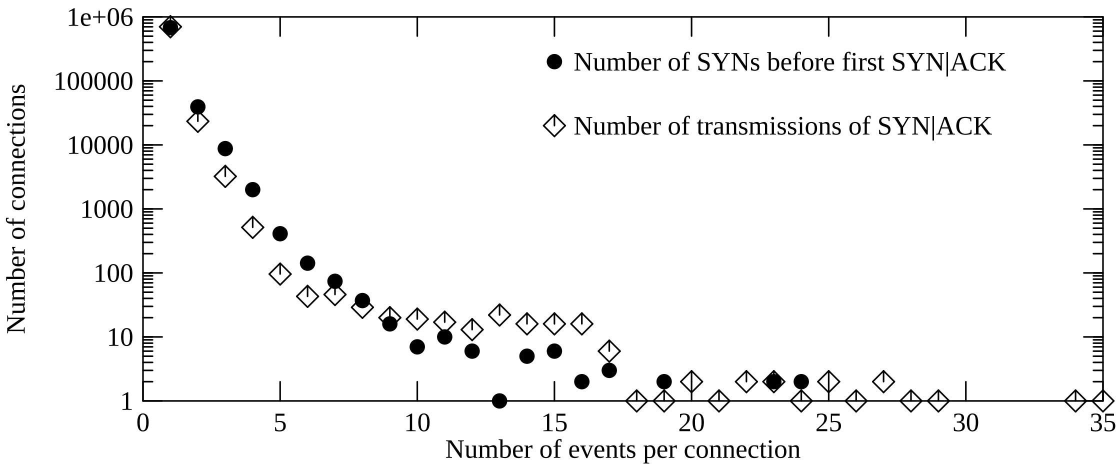


Figure 8-4: Distributions of SYN and SYN|ACK retransmission counts, Nov. 9

8.3.2. Data retransmissions

Examining retransmissions of TCP data packets (non-SYN packets, since in most cases SYNs do not carry data) takes more effort. One has to look for overlaps in the sequence number ranges of various packets, rather than simply looking for multiple occurrences of a particular SYN. Also, while SYN retransmissions normally appear in sequence, TCP data retransmissions may appear out of sequence.

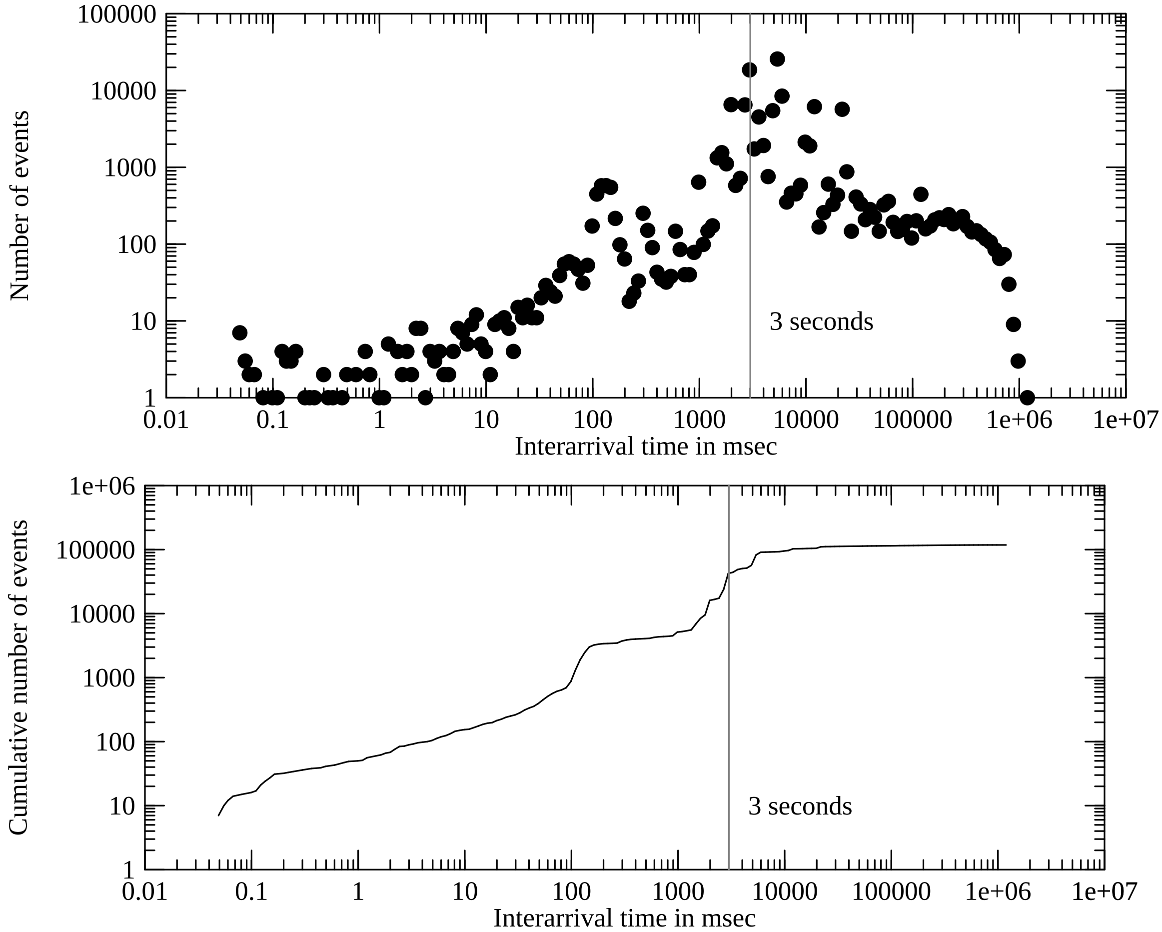


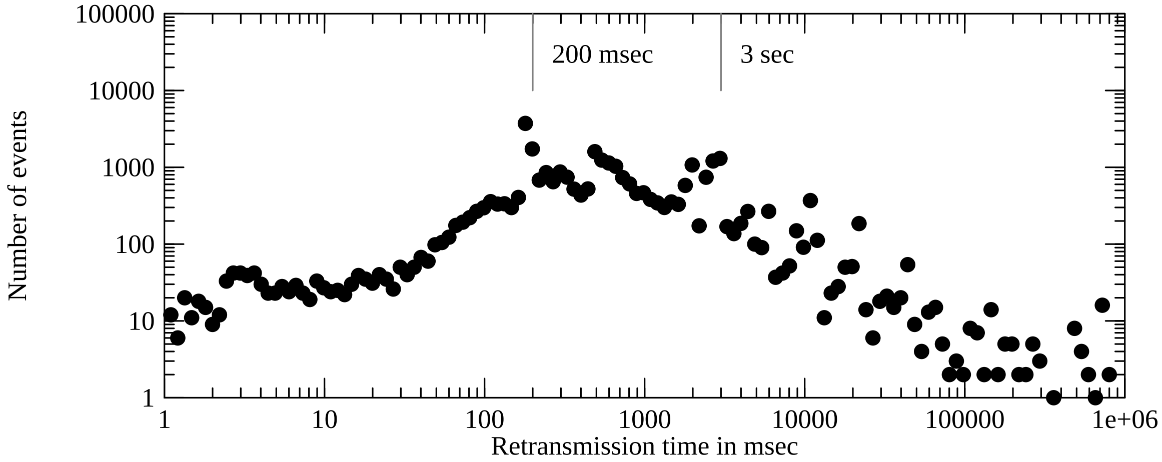
Figure 8-5: Distributions of SYN retransmit interarrival times, Nov. 9

My procedure for analyzing traces for data retransmissions involved these steps:

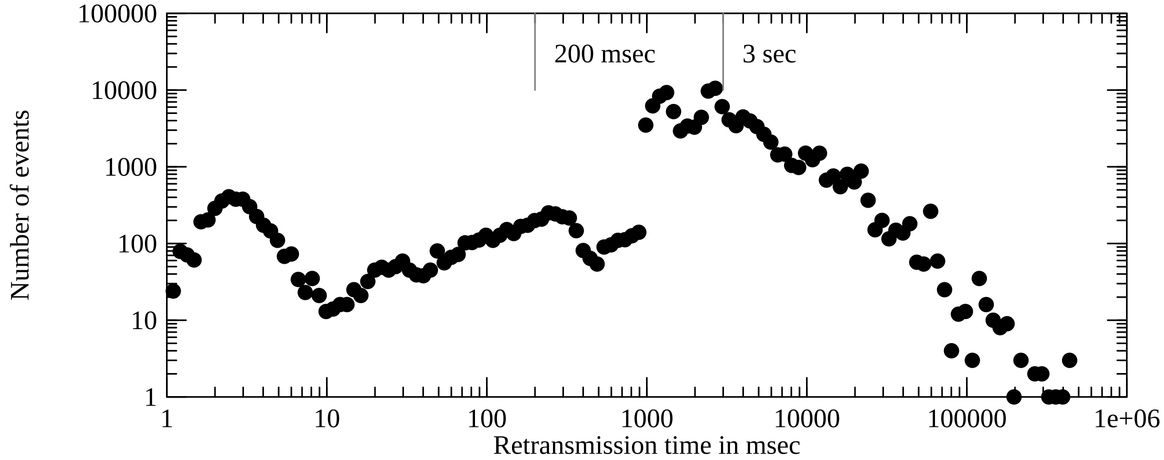
1. Generate a formatted trace, without SYN or RST packets. (I included FINs because they are retransmitted similarly to data packets.)
2. Extract for each packet a connection ID, consisting of source and destination host and port numbers, a timestamp, the starting sequence number, and the ending sequence number.
3. Sort the result using the connection ID as the major sort key, and the starting sequence number as the minor sort key.
4. Sequentially read through the sorted sequence, looking for adjacent records with identical IDs and overlapping sequence ranges. When such adjacencies are found, generate a record including the difference between the timestamps.

The records generated in the last step can be further analyzed to remove multiple retransmissions of the same sequence number range, although the algorithm used for this is not perfect. Many of the duplicate ‘‘retransmissions’’ by the server were actually ‘‘persist timer’’ transmissions, used when probing a receiver’s zero window. A missing timeout in the BSD code can cause this activity to last forever, if the client disappears from the Internet.

Figure 8-6 shows the distribution of retransmission times (times between the first and second transmission of a sequence range). The upper graph shows results for retransmissions by client



Client-generated retransmissions



Server-generated retransmissions

Figure 8-6: Distributions of data retransmission times, Nov. 9

hosts; the lower graph shows retransmissions by the HTTP server hosts. Altogether, the client distribution shows 31934 unique retransmissions events (36179, including duplicates); the server distribution shows 120423 unique retransmissions (155682, including duplicates). Retransmitted data packets represent just 0.6% of the total client transmissions, and 2.4% of the total server transmissions.

The distributions generally have the same shape, with a few exceptions. Several features in these distributions deserve explanation:

- **The peak at around 3 msec (especially for the servers)**: This may be an artifact of the routing topology, which caused transmissions from the servers to appear twice on the local Ethernet. Most of these transmissions have been filtered out of the traces, but apparently not all of them.
- **The peak at 200 msec**: The retransmission timers in BSD-based systems are examined every 200 msec, so this is the smallest value representable. The smaller timeouts observed in these traces could be the result of the quantization in this timer, or could be due to non-BSD systems.

- **The peak at about 1.3 seconds (for the servers)**: This appears to be the normal result of the BSD retransmission timeout calculation, which allows a minimum timeout of 1 second. RFC1122 [4] recommends a smaller minimum, but the original TCP specification [15] specified this 1-second value. The additional 0.3 seconds probably represents an estimate of the variance in round-trip times. This steep peak is not seen in the client distribution, perhaps because some clients are using smaller minimum timeouts, or perhaps because the variance in packet delay through the Internet smears out the interval between transmissions.
- **The peak at 3 seconds**: This is the recommended initial value of the retransmission timer; this peak represents retransmissions that occur before the sender has been able to measure the round-trip time, or after it has decided that its current estimate is definitely wrong.

The shape of these distributions, especially for retransmissions from the clients, lends support to the conclusion drawn from figure 6-3, that typical round-trip times were on the order of 80-100 msec. The retransmission timeout should be somewhat larger than the actual round-trip time, and that is in fact what one sees in figure 8-6: peaks between 100 and 300 msec.

9. Summary and conclusions

By carefully collecting log and packet trace data during the operation of the 1994 California Election server, we obtained an immensely valuable resource for analyzing the behavior of hosts and the Internetwork. We strongly recommend that other people expecting heavy usage of Internet services collect similar logs, since we expect that our results are somewhat specific to our application.

My analysis of this data support several conclusions:

- The use of a new TCP connection for each HTTP request wastes server resources (PCB table space, CPU time). The relatively short interarrival times for requests from active clients suggests that enough locality of reference exists to justify a different HTTP design.
- Our expectation that browsers and proxies would cache inlined images proved to be wishful thinking, in many cases. Perhaps browsers should have several ‘‘Reload’’ buttons, or perhaps the HTTP protocol or HTML format should provide a way to specify cachability. ‘‘Perfect’’ caching would have eliminated a large fraction of our load.
- Our use of the Domain Name Service for crude load balancing seems to have worked, although not perfectly.
- Interarrival times for both packets and requests follow Poisson distributions at some time scales, but generally not for interarrival times much above the mean. These distributions also show interesting non-Poisson structure at very short time scales, apparently due to queueing effects. For use in server capacity planning, however, Poisson distributions may be sufficiently predictive.

In this paper I have only scratched the surface of the analyses that could be applied to this data, and I hope other researchers will be inspired to find new ways of looking at similar experiments.

Acknowledgements

David Jefferson masterminded the 1994 California election service. Many other people contributed to the success of the election service, and in particular helped me obtain logs and traces; these include Paul Flaherty, Steve Glassman, Brad Horak, Richard Schedler, Stephen Stuart, Glenn Trewitt, Annie Warren, and Jason Wold. Paul Flaherty and Kathy Richardson helped to proofread various drafts of this report.

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] T. Berners-Lee, R. T. Fielding, and H. Frystyk Nielsen. *Hypertext Transfer Protocol -- HTTP/1.0*. Internet Draft draft-ietf-http-v10-spec-00.txt, IETF, March, 1995. This is a work in progress.

[3] Jean-Chrysostome Bolot. End-to-End Packet Delay and Loss Behavior in the Internet. In *Proc. SIGCOMM '93 Symposium on Communications Architectures and Protocols*, pages 289-298. San Francisco, CA, September, 1993.

[4] R. Braden. *Requirements for Internet Hosts -- Communication Layers*. RFC 1122, Internet Engineering Task Force, October, 1989.

[5] Mark E. Crovella and Azer Bestavros. *Explaining World Wide Web Traffic Self-Similarity*. Technical Report TR-95-015, Computer Science Department, Boston University, August, 1995.

[6] Steven Glassman. A Caching Relay for the World Wide Web. In *Proceedings of the First International World-Wide Web Conference*, pages 69-76. Geneva, May, 1994.

[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] Raj Jain and Shawn Routhier. Packet Trains: Measurements and a New Model for Computer Network Traffic. *IEEE Journal on Selected Areas in Communication* SAC-4(6):986-995, September, 1986.

[9] Phil Karn and Craig Partridge. Improving Round-Trip Time Estimates in Reliable Transport Protocols. *ACM Transactions on Computer Systems* 6(4):364-373, November, 1991.

[10] Will E. Leland, Murad S. Taqqu, Walter Willinger, and Daniel V. Wilson. On the Self-Similar Nature of Ethernet Traffic. In *Proc. SIGCOMM '93 Symposium on Communications Architectures and Protocols*, pages 183-193. San Francisco, CA, September, 1993.

[11] Paul E. McKenney and Ken F. Dove. Efficient Demultiplexing of Incoming TCP Packets. In *Proc. SIGCOMM '92 Symposium on Communications Architectures and Protocols*, pages 269-279. Baltimore, MD, August, 1992.

[12] David L. Mills. Improved Algorithms for Synchronizing Computer Network Clocks. In *Proc. SIGCOMM '94 Symposium on Communications Architectures and Protocols*, pages 317-327. London, UK, August, 1994.

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

[14] Vern Paxson and Sally Floyd. Wide-Area Traffic: The Failure of Poisson Modeling. In *Proc. SIGCOMM '94 Symposium on Communications Architectures and Protocols,* pages 257-268. London, UK, August, 1994.

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

[16] Jon Postel. *Internet Control Message Protocol*. RFC 792, Network Information Center, SRI International, September, 1981.

[17] Walter Willinger, Murad S. Taqqu, Robert Sherman, and Daniel V. Wilson. Self-Similarity Through High-Variability: Statistical Analysis of Ethernet LAN Traffic at the Source Level. In *Proc. SIGCOMM '95 Symposium on Communications Architectures and Protocols,* pages 100-113. Cambridge, MA, August, 1995.

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.

‘‘Drip: A Schematic Drawing Interpreter.’’ **Ramsey W. Haddad.** WRL Research Report 95/1, March 1995.

‘‘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, October 1995.

‘‘The Predictability of Branches in Libraries.’’ **Brad Calder, Dirk Grunwald, and Amitabh Srivastava.** WRL Research Report 95/6, October 1995.

‘‘Shared Memory Consistency Models: A Tutorial.’’ **Sarita V. Adve, Kourosh Gharachorloo.** WRL Research Report 95/7, September 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.

‘‘Characterization of Organic Illumination Systems.’’ **Bill Hamburgen, Jeff Mogul, Brian Reid, Alan Eustace, Richard Swan, Mary Jo Doherty, and Joel Bartlett.** WRL Technical Note TN-13, April 1989.

‘‘Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers.’’ **Norman P. Jouppi.** WRL Technical Note TN-14, March 1990.

‘‘Limits of Instruction-Level Parallelism.’’ **David W. Wall.** WRL Technical Note TN-15, December 1990.

‘‘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.

‘‘Piecewise Linear Models for Rsim.’’ **Russell Kao, Mark Horowitz.** WRL Technical Note TN-40, December 1993.

‘‘Speculative Execution and Instruction-Level Parallelism.’’ **David W. Wall.** WRL Technical Note TN-42, March 1994.

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

‘‘ATOM: A Flexible Interface for Building High Performance Program Analysis Tools.’’ **Alan Eustace and Amitabh Srivastava.** WRL Technical Note TN-44, July 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.

‘‘The Predictability of Libraries.’’ **Brad Calder, Dirk Grunwald, Amitabh Srivastava.** WRL Technical Note TN-50, July 1995.

WRL Research Reports and Technical Notes are available on the World Wide Web, from `http://www.research.digital.com/wrl/techreports/index.html`.