MAY 1995

WRL
Research Report 95/4

The Case for Persistent-Connection HTTP

Jeffrey C. Mogul

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

The Case for Persistent-Connection HTTP

Jeffrey C. Mogul

May, 1995

Abstract

The success of the World-Wide Web is largely due to the simplicity, hence ease of implementation, of the Hypertext Transfer Protocol (HTTP). HTTP, however, makes inefficient use of network and server resources, and adds unnecessary latencies, by creating a new TCP connection for each request. Modifications to HTTP have been proposed that would transport multiple requests over each TCP connection. These modifications have led to debate over their actual impact on users, on servers, and on the network. This paper reports the results of log-driven simulations of several variants of the proposed modifications, which demonstrate the value of persistent connections.

This Research Report is an expanded version of a paper to appear in the *Proceedings of the SIGCOMM '95 Conference on Communications Architectures and Protocols*.

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

Table of Contents

List of Figures

1. Introduction

People use the World Wide Web because it gives quick and easy access to a tremendous variety of information in remote locations. Users do not like to wait for their results; they tend to avoid or complain about Web pages that take a long time to retrieve. Users care about Web latency.

Perceived latency comes from several sources. Web servers can take a long time to process a request, especially if they are overloaded or have slow disks. Web clients can add delay if they do not quickly parse the retrieved data and display it for the user. Latency caused by client or server slowness can in principle be solved simply by buying a faster computer, or faster disks, or more memory.

The main contributor to Web latency, however, is network communication. The Web is useful precisely because it provides remote access, and transmission of data across a distance takes time. Some of this delay depends on bandwidth; you can reduce this delay by buying a higher-bandwidth link. But much of the latency seen by Web users comes from propagation delay, and you cannot improve propagation delay (past a certain point) no matter how much money you have. While caching can help, many Web access are ‘‘compulsory misses.’’

If we cannot increase the speed of light, we should at least minimize the number of network round-trips required for an interaction. The Hypertext Transfer Protocol (HTTP) [3], as it is currently used in the Web, incurs many more round trips than necessary (see section 2).

Several researchers have proposed modifying HTTP to eliminate unnecessary network round-trips [21, 27]. Some people have questioned the impact of these proposals on network, server, and client performance. This paper reports on simulation experiments, driven by traces collected from an extremely busy Web server, that support the proposed HTTP modifications. According to these simulations, the modifications will improve user’s perceived performance, network loading, and server resource utilization.

The paper begins with an overview of HTTP (section 2) and an analysis of its flaws (section 3). Section 4 describes the proposed HTTP modifications, and section 5 describes some of the potential design issues of the modified protocol. Section 7 describes the design of the simulation experiments, and section 8 describes the results.

2. Overview of the HTTP protocol

The HTTP protocol [1, 3] is layered over a reliable bidirectional byte stream, normally TCP [23]. Each HTTP interaction consists of a request sent from the client to the server, followed by a response sent from the server to the client. Request and response parameters are expressed in a simple ASCII format (although HTTP may convey non-ASCII data).

An HTTP request includes several elements: a *method* such as GET, PUT, POST, etc.; a Uniform Resource Locator (URL); a set of Hypertext Request (HTRQ) headers, with which the client specifies things such as the kinds of documents it is willing to accept, authentication information, etc; and an optional Data field, used with certain methods (such as PUT).

The server parses the request, and takes action according to the specified method. It then sends a response to the client, including (1) a status code to indicate if the request succeeded, or if not, why not; (2) a set of object headers, meta-information about the ‘‘object’’ returned by the server; and (3) a Data field, containing the file requested, or the output generated by a server-side script.

URLs may refer to numerous document types, but the primary format is the Hypertext Markup Language (HTML) [2]. HTML supports the use of hyperlinks (links to other documents). HTML also supports the use of inlined images, URLs referring to digitized images (usually in the Graphics Interchange Format (GIF) [7] or JPEG format), which should be displayed along with the text of the HTML file by the user’s browser. For example, if an HTML page includes a corporate logo and a photograph of the company’s president, this would be encoded as two inlined images. The browser would therefore make three HTTP requests to retrieve the HTML page and the two images.

3. Analysis of HTTP’s inefficiencies

I now analyze the way that the interaction between HTTP clients and servers appears on the network, with emphasis on how this affects latency.

Figure 1 depicts the exchanges at the beginning of a typical interaction, the retrieval of an HTML document with at least one uncached inlined image. In this figure, time runs down the page, and long diagonal arrows show packets sent from client to server or vice versa. These arrows are marked with TCP packet types; note that most of the packets carry acknowledgements, but the packets marked ACK carry *only* an acknowledgement and no new data. FIN and SYN packets in this example never carry data, although in principle they sometimes could.

Shorter, vertical arrows show local delays at either client or server; the causes of these delays are given in italics. Other client actions are shown in roman type, to the left of the Client timeline.

Also to the left of the Client timeline, horizontal dotted lines show the ‘‘mandatory’’ round trip times (RTTs) through the network, imposed by the combination of the HTTP and TCP protocols. These mandatory round-trips result from the dependencies between various packet exchanges, marked with solid arrows. The packets shown with gray arrows are required by the TCP protocol, but do not directly affect latency because the receiver is not required to wait for them before proceeding with other activity.

The mandatory round trips are:

1. The client opens the TCP connection, resulting in an exchange of SYN packets as part of TCP’s three-way handshake procedure.
2. The client transmits an HTTP request to the server; the server may have to read from its disk to fulfill the request, and then transmits the response to the client. In this example, we assume that the response is small enough to fit into a single data packet, although in practice it might not. The server then closes the TCP connection, although usually the client need not wait for the connection termination before continuing.

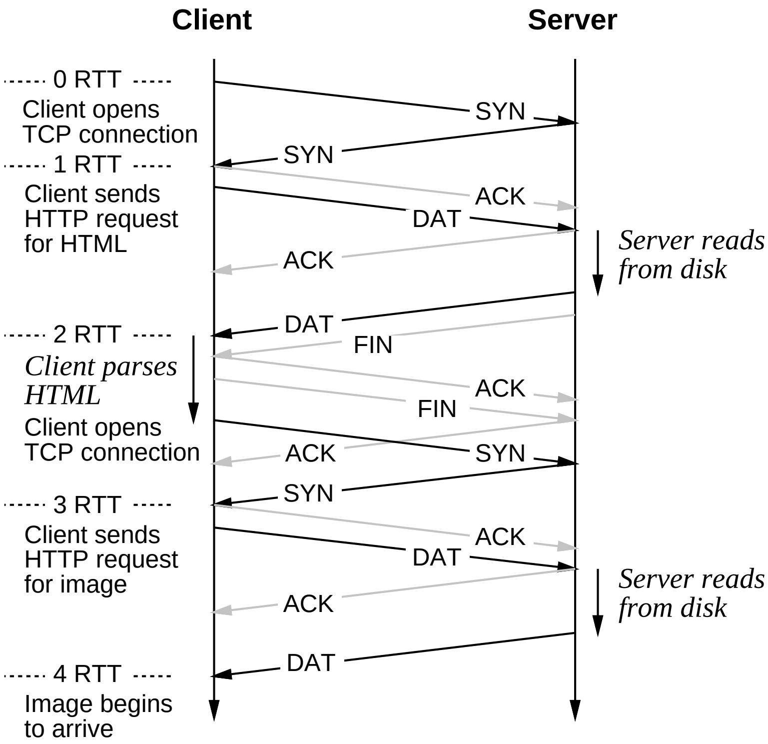


Figure 1: Packet exchanges and round-trip times for HTTP

3. After parsing the returned HTML document to extract the URLs for inlined images, the client opens a new TCP connection to the server, resulting in another exchange of SYN packets.

4. The client again transmits an HTTP request, this time for the first inlined image. The server obtains the image file, and starts transmitting it to the client.

Therefore, the earliest time at which the client could start displaying the first inlined image would be four network round-trip times after the user requested the document. Each additional inlined image requires at least two further round trips. In practice, for documents larger than can fit into a small number of packets, additional delays will be encountered.

3.1. Other inefficiencies

In addition to requiring at least two network round trips per document or inlined image, the HTTP protocol as currently used has other inefficiencies.

Because the client sets up a new TCP connection for each HTTP request, there are costs in addition to network latencies:

- Connection setup requires a certain amount of processing overhead at both the server and the client. This typically includes allocating new port numbers and resources, and creating the appropriate data structures. Connection teardown also requires some processing time, although perhaps not as much.

- The TCP connections may be active for only a few seconds, but the TCP specification requires that the host that closes the connection remember certain per-connection information for four minutes [23] (Many implementations violate this specification and use a much shorter timer.) A busy server could end up with its tables full of connections in this ‘‘TIME_WAIT’’ state, either leaving no room for new connections, or perhaps imposing excessive connection table management costs.

Current HTTP practice also means that most of these TCP connections carry only a few thousand bytes of data. I looked at retrieval size distributions for two different servers. In one, the mean size of 200,000 retrievals was 12,925 bytes, with a median of 1,770 bytes (ignoring 12,727 zero-length retrievals, the mean was 13,767 bytes and the median was 1,946 bytes). In the other, the mean of 1,491,876 retrievals was 2,394 bytes and the median 958 bytes (ignoring 83,406 zero-length retrievals, the mean was 2,535 bytes, the median 1,025 bytes). In the first sample, 45% of the retrievals were for GIF files; the second sample included more than 70% GIF files. The increasing use of JPEG images will tend to reduce image sizes.

TCP does not fully utilize the available network bandwidth for the first few round-trips of a connection. This is because modern TCP implementations use a technique called *slow-start* [13] to avoid network congestion. The slow-start approach requires the TCP sender to open its ‘‘congestion window’’ gradually, doubling the number of packets each round-trip time. TCP does not reach full throughput until the effective window size is at least the product of the round-trip delay and the available network bandwidth. This means that slow-start restricts TCP throughput, which is good for congestion avoidance but bad for short-connection completion latency. A long-distance TCP connection may have to transfer tens of thousands of bytes before achieving full bandwidth.

4. Proposed HTTP modifications

The simplest change proposed for the HTTP protocol is to use one TCP connection for multiple requests. These requests could be for both inlined images and independent Web pages. A client would open an HTTP connection to a server, and then send requests along this connection whenever it wishes. The server would send responses in the opposite direction.

This ‘‘persistent-connection’’ HTTP (P-HTTP) avoids most of the unnecessary round trips in the current HTTP protocol. For example, once a client has retrieved an HTML file, it may generate requests for all the inlined images and send them along the already-open TCP connection, without waiting for a new connection establishment handshake, and without first waiting for the responses to any of the individual requests. We call this ‘‘pipelining.’’ Figure 2 shows the timeline for a simple, non-pipelined example.

HTTP allows the server to mark the end of a response in one of several ways, including simply closing the connection. In P-HTTP, the server would use one of the other mechanisms, either sending a ‘‘Content-length’’ header before the data, or transmitting a special delimiter after the data.

While a client is actively using a server, normally neither end would close the TCP connection. Idle TCP connections, however, consume end-host resources, and so either end may choose to close the connection at any point. One would expect a client to close a connection only when it shifts its attention to a new server, although it might maintain connections to a few

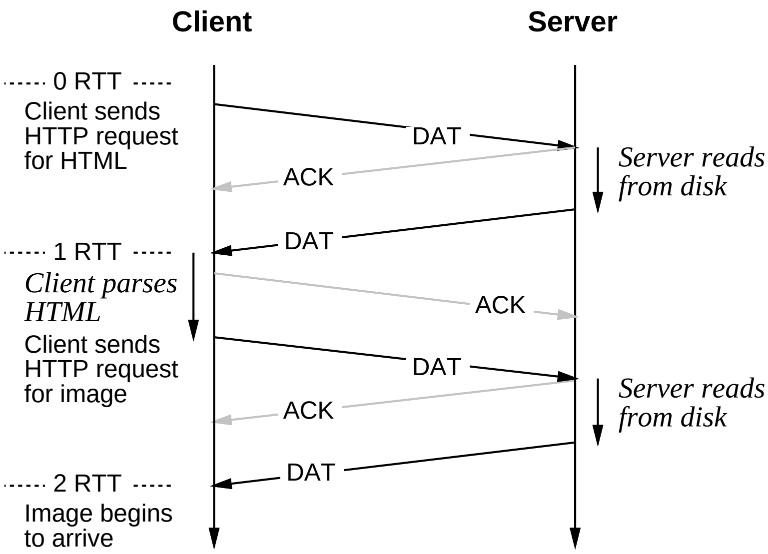


Figure 2: Packet exchanges and round-trip times for a P-HTTP interaction

servers. A client might also be ‘‘helpful’’ and close its connections after a long idle period. A client would not close a TCP connection while an HTTP request is in progress, unless the user gets bored with a slow server.

A server, however, cannot easily control the number of clients that may want to use it. Therefore, servers may have to close idle TCP connections to maintain sufficient resources for processing new requests. For example, a server may run out of TCP connection descriptors, or may run out of processes or threads for managing individual connections. When this happens, a server would close one or more idle TCP connections. One might expect a ‘‘least-recently used’’ (LRU) policy to work well. A server might also close connections that have been idle for more than a given ‘‘idle timeout,’’ in order to maintain a pool of available resources.

A server would not close a connection in the middle of processing an HTTP request. However, a request may have been transmitted by the client but not yet received when the server decides to close the connection. Or, the server may decide that the client has failed, and time out a connection with a request in progress. In any event, clients must be prepared for TCP connections to disappear at arbitrary times, and must be able to re-establish the connection and retry the HTTP request. A prematurely closed connection should not be treated as an error; an error would only be signalled if the attempt to re-establish the connection fails.

4.1. Protocol negotiation

Since millions of HTTP clients and tens of thousands of HTTP servers are already in use, it would not be feasible to insist on a globally instantaneous transition from the current HTTP protocol to P-HTTP. Neither would it be practical to run the two protocols in parallel, since this would limit the range of information available to the two communities. We would like P-HTTP servers to be usable by current-HTTP clients.

We would also like current-HTTP servers to be usable by P-HTTP clients. One could define the modified HTTP so that when a P-HTTP client contacts a server, it first attempts to use P-HTTP protocol; if that fails, it then falls back on the current HTTP protocol. This adds an extra network round-trip, and seems wasteful.

P-HTTP clients instead can use an existing HTTP design feature that requires a server to ignore HTRQ fields it does not understand. A client would send its first HTTP request using one of these fields to indicate that it speaks the P-HTTP protocol. A current-HTTP server would simply ignore this field and close the TCP connection after responding. A P-HTTP server would instead leave the connection open, and indicate in its reply headers that it speaks the modified protocol.

4.2. Implementation status

We have already published a study of an experimental implementation of the P-HTTP protocol [21]. In that paper, we showed that P-HTTP required only minor modifications to existing client and server software and that the negotiation mechanism worked effectively. The modified protocol yielded significantly lower retrieval latencies than HTTP, over both WAN and LAN networks. Since this implementation has not yet been widely adopted, however, we were unable to determine how its large-scale use would affect server and network loading.

5. Design issues

A number of concerns have been raised regarding P-HTTP. Some relate to the feasibility of the proposal; others simply reflect the need to choose parameters appropriately. Many of these issues were raised in electronic mail by members of the IETF working group on HTTP; these messages are available in an archive [12].

The first two issues discussed in this section relate to the correctness of the modified protocol; the rest address its performance.

5.1. Effects on reliability

Several reviewers have mistakenly suggested that allowing the server to close TCP connections at will could impair reliability. The proposed protocol does not allow the server to close connections arbitrarily; a connection may only be closed after the server has finished responding to one request and before it has begun to act on a subsequent request. Because the act of closing a TCP connection is serialized with the transmission of any data by server, the client is guaranteed to receive any response sent before the server closes the connection.

A race may occur between the client's transmission of a new request, and the server's termination of the TCP connection. In this case, the client will see the connection closed without receiving a response. Therefore, the client will be fully aware that the transmitted request was not received, and can simply re-open the connection and retransmit the request.

Similarly, since the server will not have acted on the request, this protocol is safe to use even with non-idempotent operations, such as the use of ‘‘forms’’ to order products.

Regardless of the protocol used, a server crash during the execution of a non-idempotent operation could potentially cause an inconsistency. The cure for this is not to complicate the network protocol, but rather to insist that the server commit such operations to stable storage before responding. The NFS specification [26] imposes the same requirement.

5.2. Interactions with current proxy servers

Many users reach the Web via ‘‘proxy’’ servers (or ‘‘relays’’). A proxy server accepts HTTP requests for any URL, parses the URL to determine the actual server for that URL, makes an HTTP request to that server, obtains the reply, and returns the reply to the original client. This technique is used to transit ‘‘firewall’’ security barriers, and may also be used to provide centralized caching for a community of users [6, 11, 22].

Section 4.1 described a technique that allows P-HTTP systems to interoperate with HTTP systems, without adding extra round-trips. What happens to this scheme if both the client and server implement P-HTTP, but a proxy between them implements HTTP [28]? The server believes that the client wants it to hold the TCP connection open, but the proxy expects the server to terminate the reply by closing the connection. Because the negotiation between client and server is done using HTRQ fields that existing proxies must ignore, the proxy cannot know what is going on. The proxy will wait ‘‘forever’’ (probably many minutes) and the user will not be happy.

P-HTTP servers could solve this problem by using an ‘‘adaptive timeout’’ scheme, in which the server observes client behavior to discover which clients are safely able to use P-HTTP. The server would keep a list of client IP addresses; each entry would also contain an ‘‘idle timeout’’ value, initially set to a small value (such as one second). If a client requests the use of P-HTTP, the server would hold the connection open, but only for the duration of the per-client idle timeout. If a client ever transmits a second request on the same TCP connection, the server would increase the associated idle timeout from the default value to a maximum value.

Thus, a P-HTTP client reaching the server through an HTTP-only proxy would encounter 1-second additional delays[1], and would never see a reply to a second request transmitted on a given TCP connection. The client could use this lack of a second reply to realize that an HTTP-only proxy is in use, and subsequently the client would not attempt to negotiate use of P-HTTP with this server.

A P-HTTP client, whether it reaches the server through a P-HTTP proxy or not, might see the TCP connection closed ‘‘too soon,’’ but if it ever makes multiple requests in a brief interval, the server’s timeout would increase and the client would gain the full benefit of P-HTTP.

The simulation results in section 8 suggest that this approach should yield most of the benefit of P-HTTP. It may fail in actual use, however; for example, some HTTP-only proxies may forward multiple requests received on a single connection, without being able to return multiple

[1]If the proxy forwards response data as soon as it is ‘‘pushed’’ by the server, then the user would not actually perceive any extra delay. This is because P-HTTP servers always indicate the end of a response using content-length or a delimiter, so the P-HTTP client will detect the end of the response even if the proxy does not.

replies. This would trick the server into holding the connection open, but would prevent the client from receiving all the replies.

5.2.1. Possibility of proxy deadlock

Some people have suggested that the use of persistent connections may increase the probability of a deadlock between two or more proxies with overcommitted resources. This potential deadlock arises in the situation depicted in figure 3, which has several unusual (but not impossible) features:

- More than one proxy is used on the path between a client and a server.
- The proxies are used to relay in both directions; that is, each member of a set of proxies acts as both client and server with respect to other proxies.

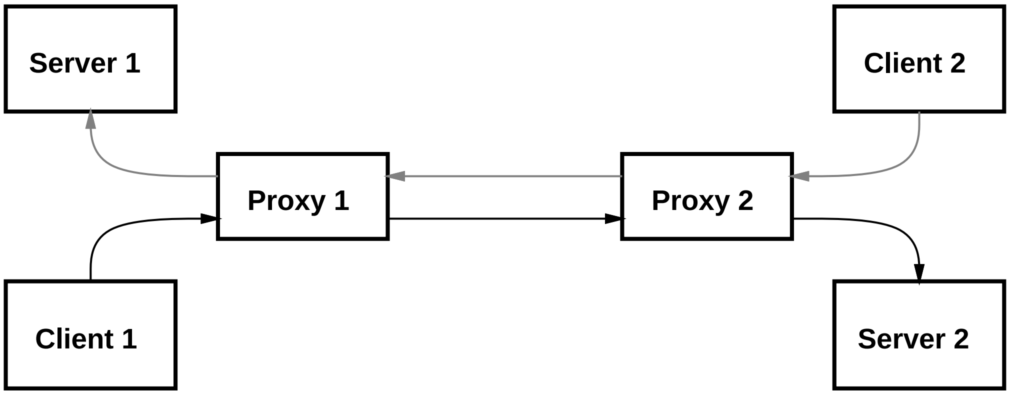


Figure 3: Situation with a potential for temporary deadlock

A deadlock could arise when both of the proxies in figure 3, or more generally each member of a cycle of proxies, has filled its quota of active TCP connections by accepting a connection from a client, and still needs to make a connection to the other in order to fulfill one of the pending requests from its clients. For example, suppose that each proxy in the figure can handle at most two TCP connections, and that they have each acccepted connections connections from their immediate clients. This leaves neither proxy with the resources to establish a connection to the other.

In order for an actual deadlock to occur, several additional conditions must exist. First, neither of the clients will asynchronously abort its connection (although a frustrated user might be expected to do this). Second, when one proxy fails to establish a connection to the other, it must wait (or continuously retry the connection attempt) instead of signalling failure. Existing proxies do not retry failed connection attempts, but immediately report them to their clients. If the specification for persistent-connection HTTP requires proxies not to retry failed TCP connections, this should avoid deadlock.

One should note that the potential for deadlock in this configuration exists even with the unmodified HTTP protocol. P-HTTP, however, could change the likelihood of resource exhaustion at proxies, since TCP connections persist for much longer with P-HTTP.

5.3. Connection lifetimes

One obvious question is whether the servers would have too many open connections in the persistent-connection model. The glib answer is ‘‘no, because a server could close an idle connection at any time’’ and so would not necessarily have more connections open than in the current model. This answer evades the somewhat harder question of whether a connection would live long enough to carry significantly more than one HTTP request, or whether the servers would be closing connections almost as fast as they do now.

Intuition suggests that locality of reference will make this work. That is, clients tend to send a number of requests to a server in relatively quick succession, and as long as the total number of clients simultaneously using a server is ‘‘small,’’ the connections should be useful for multiple HTTP requests. The simulations (see section 8) support this.

5.4. Server resource utilization

HTTP servers consume several kinds of resources, including CPU time, active connections (and associated threads or processes), and protocol control block (PCB) table space (for both open and TIME_WAIT connections). How would the persistent-connection model affect resource utilization?

If an average TCP connection carries more than one successful HTTP transaction, one would expect this to reduce server CPU time requirements. The time spent actually processing requests would probably not change, but the time spent opening and closing connections, and launching new threads or processes, would be reduced. For example, some HTTP servers create a new process for each connection. Measurements suggest that the cost of process creation accounts for a significant fraction of the total CPU time, and so persistent connections should avoid much of this cost.

Because we expect a P-HTTP server to close idle connections as needed, a busy server (one on which idle connections never last long enough to be closed by the idle timeout mechanism) will use up as many connections as the configuration allows. Therefore, the maximum number of open connections (and threads or processes) is a parameter to be set, rather than a statistic to be measured.

The choice of the idle timeout parameter (that is, how long an idle TCP connection should be allowed to exist) does not affect server performance under heavy load from many clients. It can affect server resource usage if the number of active clients is smaller than the maximum-connection parameter. This may be important if the server has other functions besides HTTP service, or if the memory used for connections and processes could be applied to better uses, such as file caching.

The number of PCB table entries required is the sum of two components: a value roughly proportional to the number of open connections (states including ESTABLISHED, CLOSING, etc.), and a value proportional to the number of connections closed in the past four minutes (TIME_WAIT connections). For example, on a server that handles 100 connections per second, each with a duration of one second, the PCB table will contain a few hundred entries related to open connections, and 24,000 TIME_WAIT entries. However, if this same server followed the persistent-connection model, with a mean of ten HTTP requests per active connection, the PCB table would contain only 2400 TIME_WAIT entries.

PCB tables may be organized in a number of different ways [16]. Depending on the data structures chosen, the huge number of TIME_WAIT entries may or may not affect the cost of looking up a PCB-table entry, which must be done once for each received TCP packet. Many existing systems derived from 4.2BSD use a linear-list PCB table [15], and so could perform quite badly under a heavy connection rate. In any case, PCB entries consume storage.

The simulation results in section 8 show that persistent-connection HTTP significantly reduces the number of PCB table entries required.

5.5. Server congestion control

An HTTP client has little information about how busy any given server might be. This means that an overloaded HTTP server can be bombarded with requests that it cannot immediately handle, leading to even greater overload and congestive collapse. (A similar problem afflicts naive implementations of NFS [14].) The server could cause the clients to slow down, somewhat, by accepting their TCP connections but not immediately processing the associated requests. This might require the server to maintain a very large number of TCP connections in the ESTABLISHED state (especially if clients attempt to use several TCP connections at once; see section 6).

Once a P-HTTP client has established a TCP connection, however, the server can automatically benefit from TCP's flow-control mechanisms, which prevent the client from sending requests faster than the server can process them. So while P-HTTP cannot limit the rate at which new clients attack an overloaded server, it does limit the rate at which any given client can make requests. The simulation results presented in section 8, which imply that even very busy HTTP servers see only a small number of distinct clients during any brief interval, suggest that controlling the per-client arrival rate should largely solve the server congestion problem.

5.6. Network resources

HTTP interactions consume network resources. Most obviously, HTTP consumes bandwidth, but IP also imposes per-packet costs on the network, and may include per-connection costs (e.g., for firewall decision-making). How would a shift to P-HTTP change consumption patterns?

The expected reduction in the number of TCP connections established would certainly reduce the number of ‘‘overhead’’ packets, and would presumably reduce the total number of packets transmitted. The reduction in header traffic may also reduce the bandwidth load on low-bandwidth links, but would probably be insignificant for high-bandwidth links.

The shift to longer-lived TCP connections should improve the congestion behavior of the network, by giving the TCP end-points better information about the state of the network. TCP senders will spend proportionately less time in the ‘‘slow-start’’ regime [13], and more time in the ‘‘congestion avoidance’’ regime. The latter is generally less likely to cause network congestion.

At the same time, a shift to longer TCP connections (hence larger congestion windows) and more rapid server responses will increase short-term bandwidth requirements, compared to current HTTP usage. In the current HTTP, requests are spaced several round-trip times apart; in P-HTTP, many requests and replies could be streamed at full network bandwidth. This may affect the behavior of the network.

5.7. User's perceived performance

The ultimate measure of the success of a modified HTTP is its effect on the user's perceived performance (UPP). Broadly, this can be expressed as the time required to retrieve and display a series of Web pages. This differs from simple retrieval latency, since it includes the cost of rendering text and images. A design that minimizes mean retrieval latency may not necessarily yield the best UPP.

For example, if a document contains both text and several inlined images, it may be possible to render the text before fully retrieving all of the images, if the user agent can discover the image ''bounding boxes'' early enough. Doing so may allow the user to start reading the text before the complete images arrive (especially if some of the images are initially off-screen). Thus, the order in which the client receives information from the server can affect UPP.

Human factors researchers have shown that users of interactive systems prefer response times below two to four seconds [25]; delays of this magnitude cause their attention to wander. Two seconds represents just 28 cross-U.S. round-trips, at the best-case RTT of about 70 msec.

Users may also be quite sensitive to high variance in UPP. Generally, users desire predictable performance [17]. That is, a user may prefer a system with a moderately high mean retrieval time and low variance, to one with lower mean retrieval time but a much higher variance. Since congestion or packet loss can increase the effective RTT to hundreds or thousands of milliseconds, this leaves HTTP very few round-trips to spare.

6. Competing and complementary approaches

Persistent-connection HTTP is not the only possible solution to the latency problem. The *NetScape* browser takes a different approach, using the existing HTTP protocol but often opening multiple connections in parallel. For example, if an HTML file includes ten inlined images, *NetScape* opens an HTTP connection to retrieve the HTML file, then might open ten more connections in parallel, to retrieve the ten image files. By parallelizing the TCP connection overheads, this approach eliminates a lot of the unnecessary latency, without requiring implementation of a new protocol.

The multi-connection approach has several drawbacks. First, it seems to increase the chances for network congestion; apparently for this reason, *NetScape* limits the number of parallel connections (a user-specifiable limit, defaulting to four). Several parallel TCP connections are more likely to self-congest than one connection.

Second, the *NetScape* approach does not allow the TCP end-points to learn the state of the network. That is, while P-HTTP eliminates the cost of slow-start after the first request in a series, *NetScape* must pay this cost for every HTML file, and for every group of parallel image retrievals.

The multi-connection approach sometimes allows *NetScape* to render the text surrounding at least the first *N* images (where *N* is the number of parallel connections) before much of the image data arrives. Some image formats include bounding-box information at the head of the file; *NetScape* can use this to render the text long before the entire images are available, thus improving UPP.

This is not the only way to discover image sizes early in the retrieval process. For example, P-HTTP could include a new method allowing the client to request a set of image bounding boxes before requesting the images. Or, the HTML format could be modified to include optional image-size information (as has been proposed for HTML version 3.0 [24]). Either alternative could provide the bounding-box information even sooner than the multi-connection approach. All such proposals have advantages and disadvantages, and are the subject of continuing debate in the IETF working group on HTTP.

Several people have suggested using Transaction TCP (T/TCP) [4, 5] to eliminate the delay associated with TCP's three-way handshake. T/TCP also reduces the number of TIME_WAIT entries by shortening the duration of the TIME_WAIT state. Therefore, T/TCP solves some of the same problems solved by P-HTTP. The use of T/TCP with unmodified HTTP (that is, one HTTP request per T/TCP connection) does not reduce the number of times that the client and server must modify their connection databases, nor does it support pipelining. Most important, T/TCP is still an ''experimental'' protocol and will not be widely implemented for many years. P-HTTP could be deployed immediately, using the existing enormous installed base of TCP implementations. If T/TCP becomes widely deployed, it should be possible to layer P-HTTP over T/TCP instead of TCP, but this change probably will not yield significant benefits.

Since P-HTTP does not change the basic nature of HTTP's mechanisms for communicating request and response information, it should be fully compatible with most of the proposed extensions to HTTP. For example, the Secure HyperText Transfer Protocol (SHTTP) [10] should work just as well with persistent connections, although we have not tested this.

7. Simulation experiment design

In order to answer some of the open questions about the performance of P-HTTP, I decided to simulate the behavior of a P-HTTP server using input streams taken from the logs of actual HTTP servers. This allowed me to explore the effect of various parameter combinations and policies. The use of actual event streams, rather than a synthetic load, should produce realistic results.

The specific open questions addressed by these simulations include:

- Do clients display sufficient locality of reference to allow each connection to carry several HTTP requests (see section 5.3)?
- Does P-HTTP reduce server resource utilization (see section 5.4)?
- Does the adaptive timeout mechanism, proposed in section 5.2 to deal with unmodified proxy servers, destroy the utility of the proposal?

The simulations were also designed to investigate how the values of several parameters, including table sizes and timeout durations, would affect performance.

The systems from which the logs were taken use the NCSA *httpd* server, version 1.3, with minor modifications to improve performance. Since this program generates a log without connection durations or fine-grained timestamps, I modified the server to generate an additional log file with the information necessary to drive the simulations. The new log includes a connection completion timestamp and the connection duration of each request. All timing information was done with a resolution of about 1 msec.

7.1. Trace data sets

I used logs from two different servers to drive the simulations. One data set came from the 1994 California Election service, and includes over 1.6 million HTTP requests in a ten-day period; the busiest 24-hour period includes almost 1 million requests. The other data set came from a large corporation's public Web site, and includes 3.4 million HTTP requests over approximately 82 days.

The election service was actually implemented as a set of three individual servers that shared a single alias in the host name space. Clients tended to load-share among the three servers. The corporate server is a single computer.

Although both data sets come from relatively busy servers, they differ in several ways. The election service was used quite intensively over just a few days. The corporate web site encountered far lower peak loads. The election service saw 24,000 distinct client addresses; the corporate server saw 134,000 clients.

Some of these client addresses represent intermediate proxies, and so aggregate requests from many different users. This should not affect the simulation, since one would see this aggregation with either HTTP or P-HTTP.

Since the two services provide different kinds of information, they saw somewhat different access patterns. Figure 4 shows the cumulative distribution of retrieval sizes (the number of bytes returned by the server for each request). The election service returned many files shorter than 100 bytes, while the corporate server provided mostly files longer than 1000 bytes.

The majority of retrievals from both servers took less than 1 second (see figure 5). However, the corporate server saw a somewhat larger fraction that took between 1 and 10 seconds. The retrievals with very short durations were made by nearby clients or proxies.

Figure 6 shows the distribution of request interarrival times for both servers. The spike in the distributions near 10 msec probably reflects the CPU-time cost to dispatch a new process for each request; the distributions for each individual server in the election service (not shown) contain almost no interarrival times shorter than this peak.

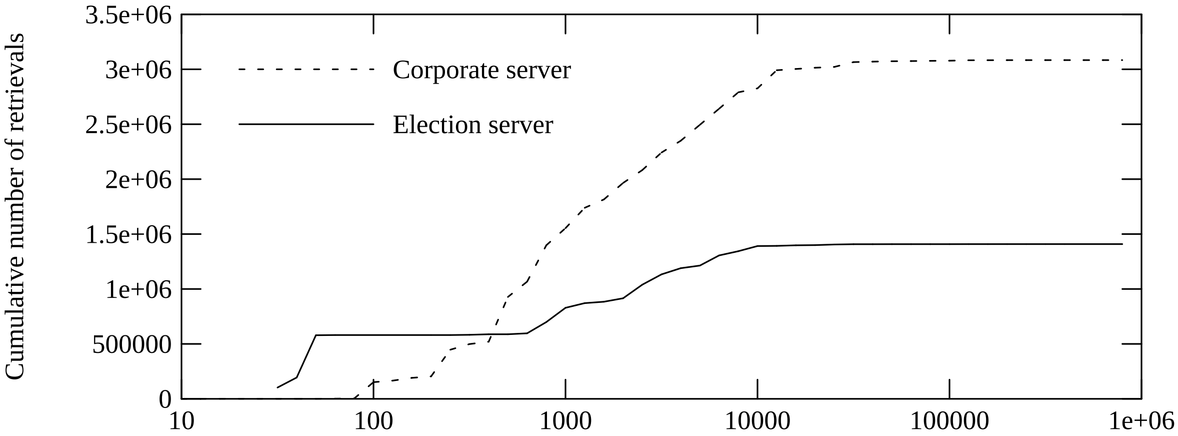


Figure 4: Cumulative distribution of retrieval sizes

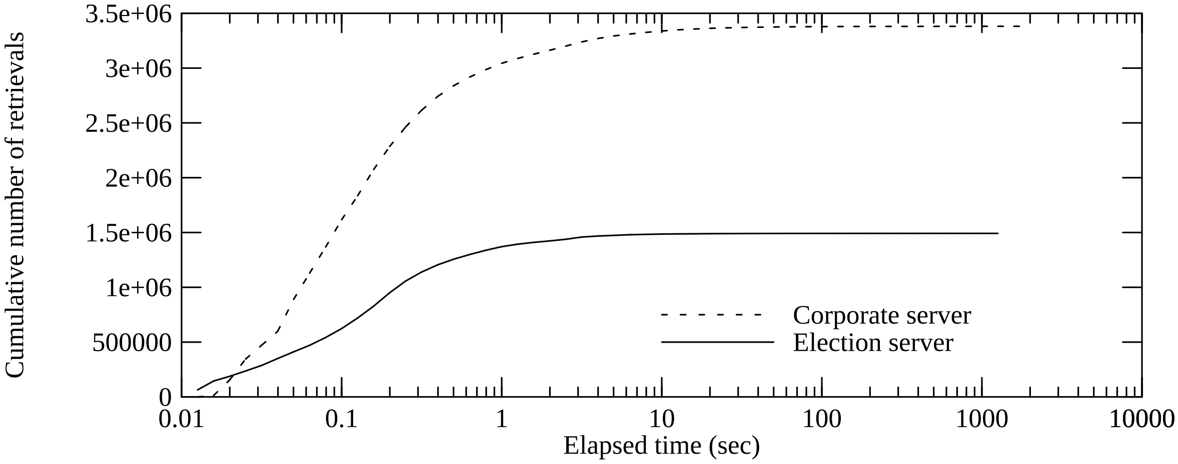


Figure 5: Cumulative distribution of connection durations

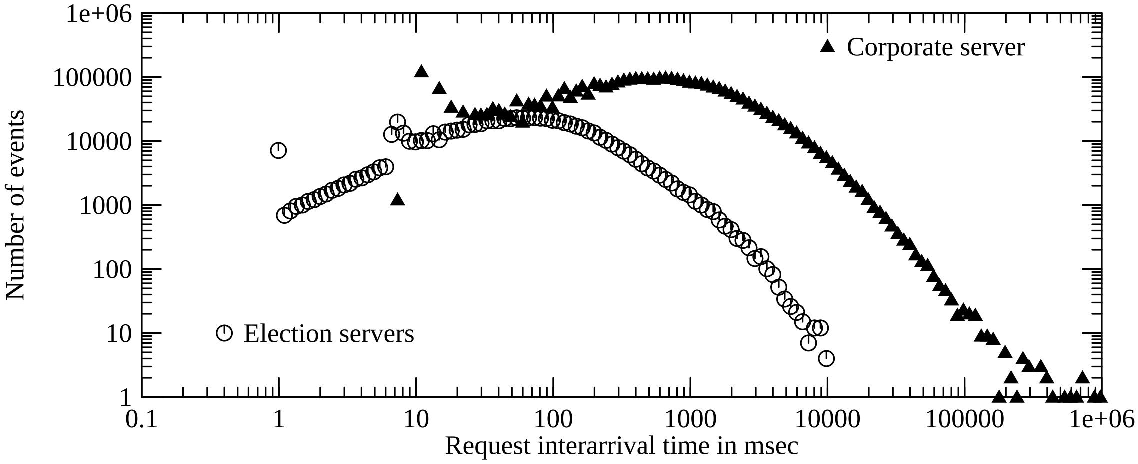


Figure 6: HTTP request interarrival times, as seen by server

7.1.1. Limitations of the traces used

Traces taken from just two HTTP servers clearly do not necessarily capture the full range of possible behavior. It may be that other servers see much less (or much more) locality of reference, or that as the client population scales up, the ‘‘working set’’ of simultaneously active clients seen by a server could increase beyond the number of available connections. Because, however, the Election service was designed to attract many clients during a brief period, its traces may come closer to representing the busy servers of the future than would traces from most other contemporary servers.

Lightly-used servers should see much higher locality of reference, since they will tend to have far few simultaneously active clients. Note also that the corporate server was lightly used during many periods; as figure 6 shows, a substantial number of its request arrivals were separated by more than 10 seconds (10,000 msec).

These simulations do not directly address the complete behavior of individual clients, since the traces were made at the servers. One would have to gather client-side traces from a large set of clients in order to prove that the typical client focusses its attention on a small set of servers for periods of several seconds or longer. However, from simple observations of how people actually use the Web, one could quite reasonably infer this to be the case.

Nor do these simulations directly address how different client caching strategies would affect the results. Since the traces were generated by real clients, most of which presumably were using caches, these simulations do reflect the use of normal client-side caching techniques.

7.2. Simulator overview

The simulator, a simple program consisting of about 1400 lines of C code, models the relevant behavior of a P-HTTP server, tracking several kinds of server state. It maintains a count of the number of open connections, and simulates the server’s PCB table, so that it can keep track of the number of TIME_WAIT entries. It can also maintain an ‘‘adaptive timeout’’ database of any given size.

Note that the simulator does not simulate the network or the clients, nor does it simulate the HTTP or TCP protocols. It simulates only the connection (‘‘session-layer’’) behavior of the server. Client and network behavior is provided by the traces of HTTP accesses, and so any effect that the modified protocol might have on client or network behavior is not modelled. Also, since the simulator sees requests arrive at the same spacing as in the original HTTP-based trace, these simulations do not account for the ‘‘pipelining’’ made possible by P-HTTP; they underestimate the potential locality of reference.

The simulator starts by parsing the log files. Each log file record is turned into a pair of event records, one for the connection-open event and one for the connection-close event. An event record contains a timestamp, an IP address, a unique connection ID-number, and flag indicating ‘‘open’’ or ‘‘close.’’ The connection-open timestamp is derived from the connection-close timestamp and connection duration, both found in the log file. After the file is parsed, the simulator sorts the event records into timestamp order.

The simulator then goes through the event records, in time-sequence order. If it is simulating an HTTP server (that is, one request per connection), it simply processes the connection-open and connection-close events verbatim, maintaining the PCB table and removing TIME_WAIT entries as they reach the 2*MSL age.

If the program is simulating a P-HTTP server, it must do more work. For a connection-open event, it checks to see if a connection to the specified IP address is already open; if so, it simply updates its statistics counters. (Since the server logs cannot record the identity of the actual client process, I assume that each client host has only one process making HTTP requests. This assumption is safe for single-user and proxy clients, but is excessively liberal for busy timesharing clients. However, I know of no way to correct for this effect.)

If there is no currently-open connection, the simulated server then checks to see if it has any free connection slots (the maximum number of simultaneous connections is a parameter of the simulation). If so, it simply creates a new ESTABLISHED record in the PCB table. Otherwise, it must make room by closing an idle connection. (The simulated server closes the least-recently used connection; this replacement policy has obvious attractions, but I have not investigated other possible policies.) If no idle connection is available, the new connection is rejected.

During a simulation of a P-HTTP server, a connection-close event causes the connection to be marked as idle, but leaves it in the ESTABLISHED TCP state.

After each event record is processed, the simulator looks for connections that have been idle longer than the specified idle-timeout parameter; these are moved into the TIME_WAIT state. The simulator also looks for connections that have been in the TIME_WAIT state for the full 2*MSL waiting period, and removes them entirely from the PCB table.

If the simulation includes the adaptive timeout mechanism (see section 5.2), the simulator maintains a table listing the N most recently active clients (N is another parameter). The table entries include the idle-timeout values for each host. When a client is first seen, it is entered into this table with a minimal timeout value. If a subsequent request arrives from the client before the connection times out, the simulator increases the client's idle-timeout to the maximum value.

7.3. Summary of simulation parameters

The simulator allows specification of these parameters:

- **P-HTTP mode**: controls whether the simulated server uses the HTTP or P-HTTP protocol.
- **Maximum number of open connections**
- **Idle-timeout**: in adaptive-timeout mode, this is maximum idle timeout.
- **2*MSL timeout**: allows simulation of non-standard timeouts for the TIME_WAIT state.
- **Adaptive-timeout table size**: the number of entries in the recently-active client list, or zero to disable the adaptive-timeout mechanism.

- **Initial idle-timeout**: in adaptive-timeout mode, the idle timeout used for clients not known to be using P-HTTP.

The simulator reports a set of statistics for each trial, including the total number of HTTP requests seen, the number of requests refused because too many were already open, the total number of TCP connections created, and the maximum number of simultaneously open connections. The simulator also reports the maximum number of TIME_WAIT entries in the PCB table, and the maximum number of PCB table entries (including both TIME_WAIT and OPEN, but not any other TCP states).

For P-HTTP simulations, the statistics also include the number times a request arrived for a TCP connection that was already open, the number of times a connection was closed to make room for a new one, and the number of connections closed because of idle-timeouts.

7.4. Validation

Does the simulation accurately model reality? I could verify the simulation of the original HTTP protocol against measurements made on the election servers, since those servers logged the contents of their PCB tables every 15 minutes. (I do not have similar logs for the corporate server.)

Figure 7 shows how the simulated and actual values compare, over a busy two-day period for the election service. The curves show instantaneous values sampled every fifteen minutes (of either real or simulated time); I did not try to compute the mean (or maximum) value over each sample interval.

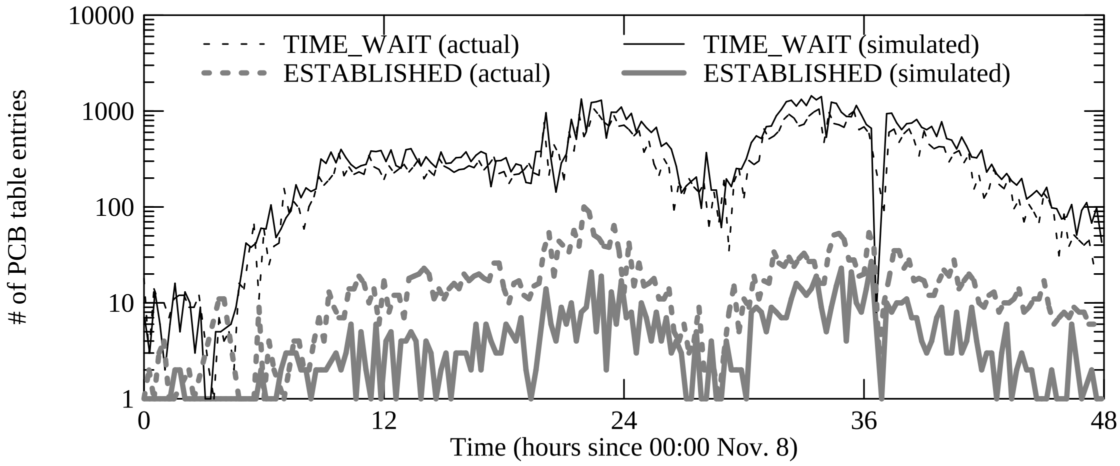


Figure 7: Comparison of simulated and actual PCB states (election service)

This simulation set the 2*MSL timeout (for the TIME_WAIT state) to 60 seconds, the value used in the operating system actually running the election service. The simulated number of TIME_WAIT entries in the PCB table agrees quite closely with the measured value. In most cases, the simulation overestimates the actual number of TIME_WAIT entries slightly (20% or less). This may be due to measurement bias on the actual system, since CPU contention may have delayed the logging of the PCB table contents during periods of heavy load.

The simulator appears to underestimate the number of ESTABLISHED (i.e, open) connections by a much wider margin. Some of the ESTABLISHED TCP connections counted in the actual measurements were not HTTP server connections (for the period shown in figure 7, there were about 5 non-HTTP connections counted in each sample), but this cannot account for the factor of two or three discrepancy at some points. In fact, many connections persisted in the ESTABLISHED state longer than the server logs indicate. The server writes its log record before closing the connection, so the logged connection durations failed to include the final network round-trip. This discrepancy does bias the simulation results, but there is no reliable way to repair the logs retroactively.

Figure 8 shows how varying the 2*MSL timeout value affects the simulated number of TIME_WAIT entries.

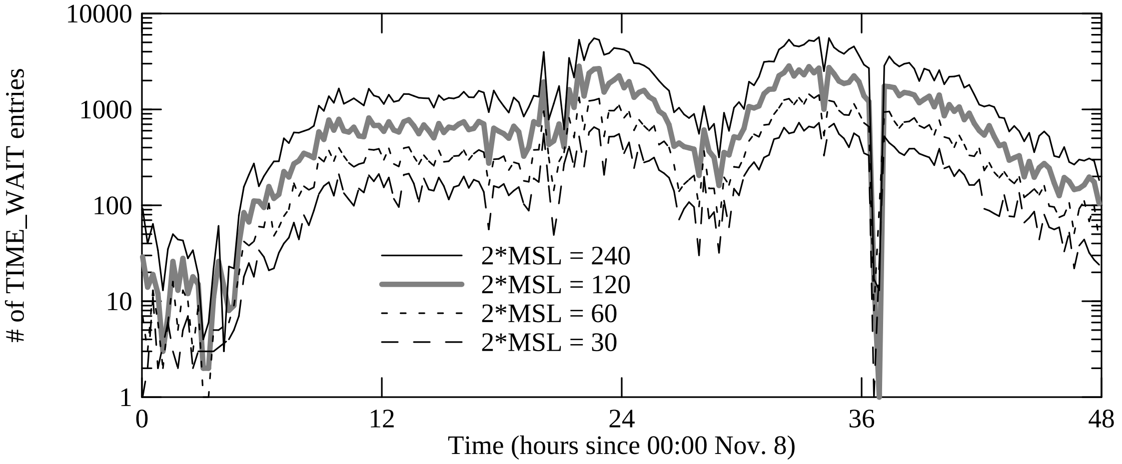


Figure 8: Effect of varying 2*MSL timer on number of TIME_WAIT entries

8. Simulation results

The first simulations compare the behavior of HTTP and several configurations of P-HTTP, varying the maximum number of connections and the idle-timeout parameter. This set of simulations did not include adaptive idle timeouts. For all of the simulations in this section, the 2*MSL timeout is set to 240 seconds (the value specified in the standard [23]).

The simulations show that for HTTP, the maximum number of simultaneously open TCP connections was rather small: 42 connections for the election service, and 30 connections for the corporate server. This means that a P-HTTP server allowing at least this many simultaneous connections would never have to refuse a request.

8.1. Connection refusal rates

What refusal rates would result if the open-connection limit were smaller? Figure 9 shows the number of connections refused as a function of the idle-timeout parameter and the maximum-connection limit (C_{max}), for both data sets. (Since ‘‘idle timeout’’ is meaningless for HTTP, the HTTP points are arbitrarily plotted at a ‘‘timeout’’ of 0.5 seconds. Also note that for all of the figures in this section, the largest idle timeout simulated was effectively infinite, but is plotted at an *x*-axis coordinate of 2000 seconds.) With a maximum of 32 connections, P-HTTP refuses significantly fewer requests than HTTP. Presumably this is because many requests that arrive during a period when all connections are in use come from a host that already owns a connection. With a maximum of 10 connections, however, P-HTTP refuses significantly more requests than HTTP. Clearly the refused requests are not coming from hosts that already own one of the 10 open connections, but it is not clear why this makes things so much worse. At any rate, these results suggest that one needs to support at least 32 simultaneous connections, for workloads resembling the ones studied.

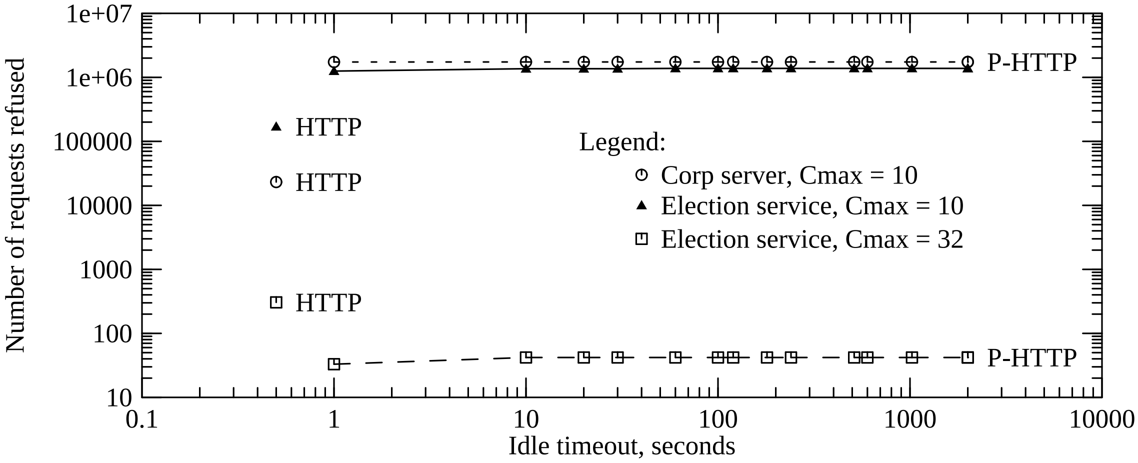


Figure 9: Requests refused due to too many simultaneous connections

8.2. Connection re-use rates

How often does the P-HTTP protocol pay off, in terms of reduced latency as seen by the user? One measure of this is the number of times a request arrives for an already-open connection. Figures 10 and 11 show the number of ‘‘open-connection hits’’ as a function of the idle-timeout parameter, for various limits on C_{max}. In each graph, the dotted line shows the theoretical best case; this is slightly less than the total number of requests made, because the first request from a client can never be a hit (and there may be other such ‘‘compulsory misses’’). The best-case hit rate for the election service is almost 99%; the best-case hit rate for the corporate server is about 95%. Even with an idle timeout of just 2 minutes, the election service would have achieved a 95% hit rate; the corporate server would have achieved 88%.

These graphs suggest that most of the benefit comes with an idle timeout of about 60 seconds. Longer timeouts yield only a negligible increase in the number of hits, except if the server can support very large numbers of active connections. Increasing C_{max} seems to increase the hit rate, although with diminishing returns once C_{max} reaches 3 or 4 times the minimum required to avoid refusing requests. Note that while P-HTTP could achieve a nearly optimal hit rate for the election service, the corporate server would not quite reach this limit, probably because of the much longer duration of the traces.

Figure 12 shows the total number of TCP connections that would be opened using P-HTTP, as a function of the idle-timeout and C_{max} parameters. (The results for the corporate server, not shown, are quite similar.) This is just the complement of the number of open-connection hits, except when C_{max} is low enough to cause some request refusals (dotted lines). For this data set, HTTP opened 1.6 million TCP connections.

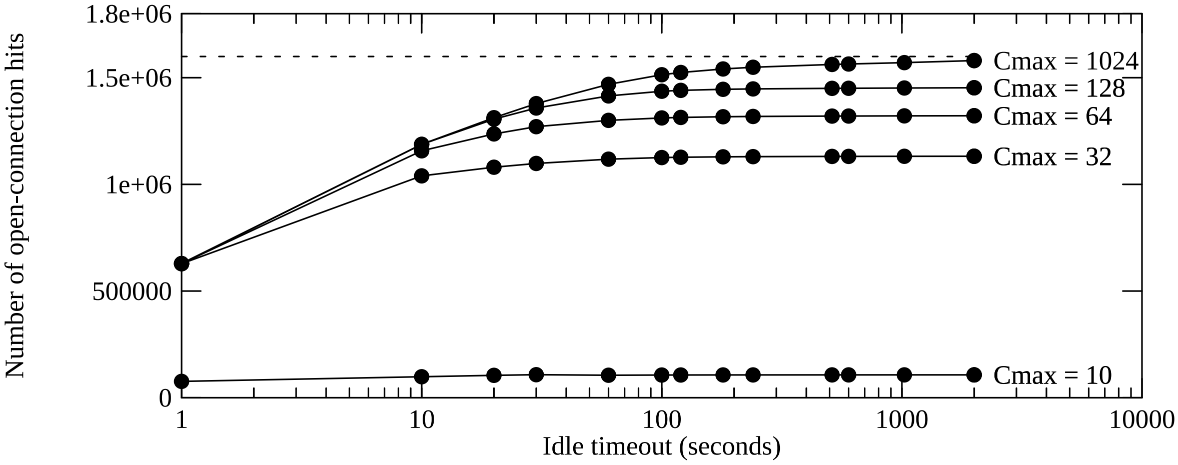


Figure 10: Number of requests arriving for already-open connections (Election service)

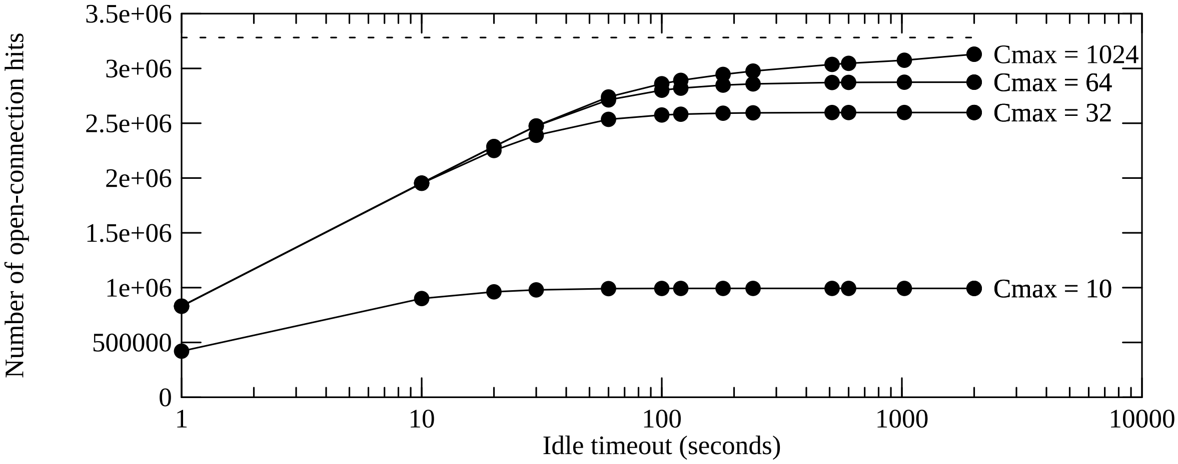


Figure 11: Number of requests arriving for already-open connections (Corporate server)

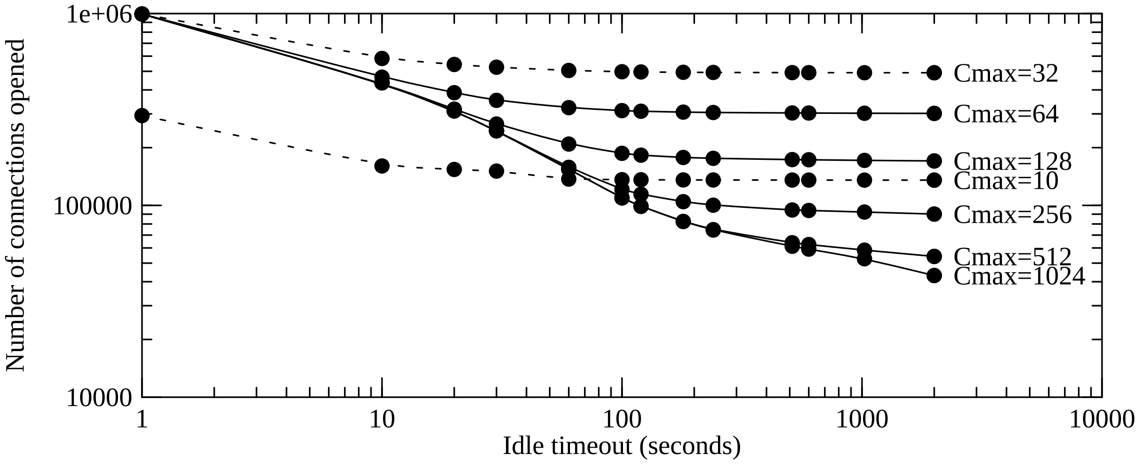


Figure 12: Number of TCP connections opened (Election service)

One can also measure the success rate of the persistent-connection approach by looking at the number of HTTP requests per TCP connection. Figure 13 shows for various parameter combinations the mean number of requests per connection for the election service. Figure 14 shows the normalized standard deviation of the number of requests (that is, the actual value of the standard deviation divided by the sample mean). Figure 15 shows the maximum number of requests per connection.

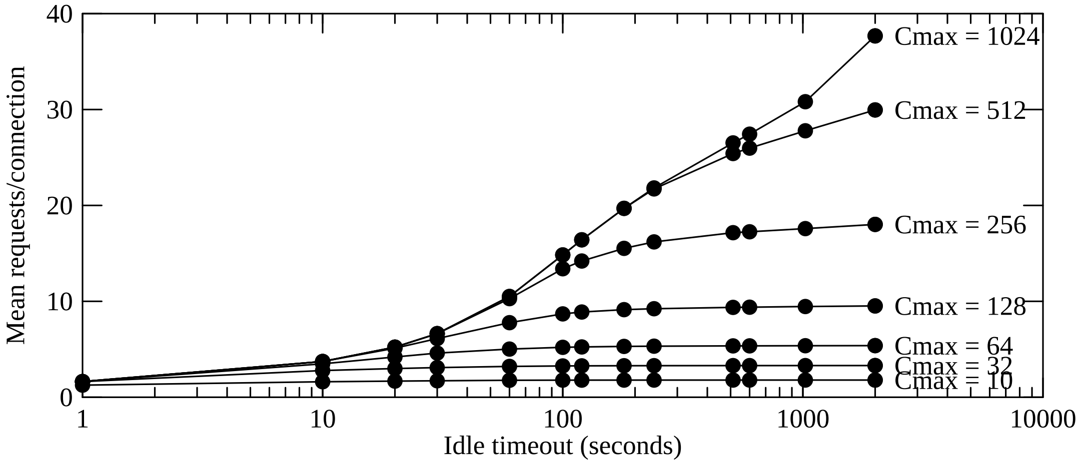
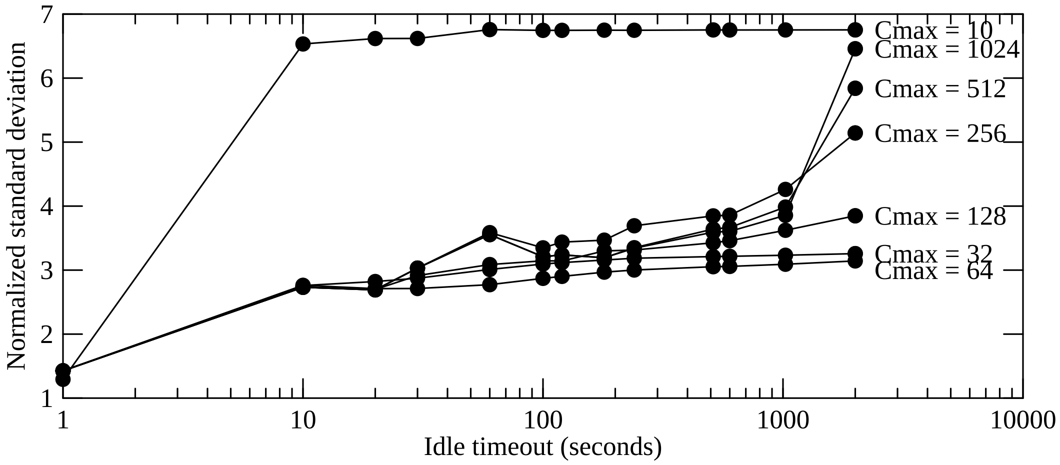


Figure 13: Mean number of HTTP requests per TCP connection (Election service)



Standard deviations normalized to means

Figure 14: Standard deviations of requests/connection (Election service)

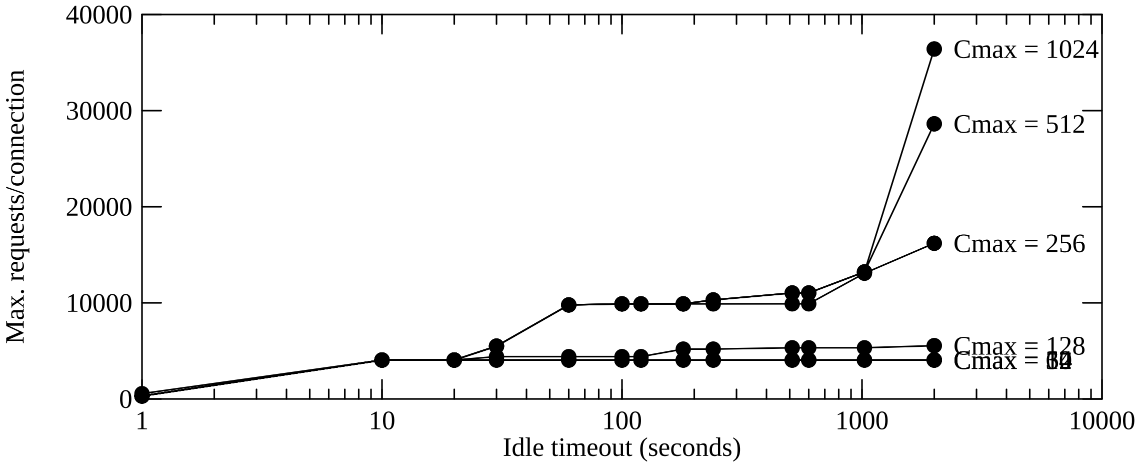


Figure 15: Maximum number of requests/connection (Election service)

From figure 13 one can see that, even with a relatively low limit on the number of active connections, on average it is not hard to satisfy ten or more HTTP requests with one TCP connection. Figure 14, however, shows that the results are quite variable, with standard deviations typical three or four times the mean. The curve for C_{max} = 10 shows a much higher standard deviation, because with so few connections available, a few clients that make frequent requests will tend to hold on to their connections, while most other clients will seldom get a chance to reuse a connection.

Since figure 15 shows that a few TCP connections, apparently those from busy proxy hosts, would handle a huge number of HTTP requests, do most connections gain from persistent connections? And do non-proxy clients see a high hit rate?

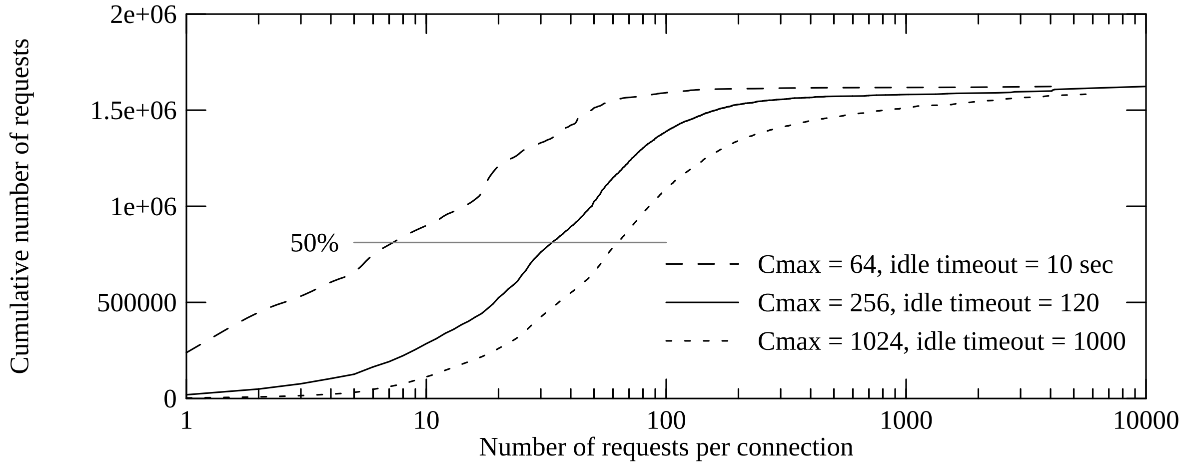


Figure 16: Cumulative number of requests/connection (Election service)

Figure 16 shows, for the election service, the cumulative number of requests per connection, for several values of the server parameters. The horizontal gray line shows where the 50th percentile lies. Over a broad range of parameter settings, a substantial number of the HTTP requests are satisfied by a connection that is reused many times.

Figures 17 through 19 show the means, standard deviations, and maxima for the corporate server. Although with this trace, P-HTTP results in much lower means than with the election service trace, the standard deviations and maxima are much higher. Some connections apparently would capture many thousands of HTTP requests, over a broad range of parameter values. Are these connections typical? The cumulative distribution in figure 20 suggests that only a small fraction of the requests are satisfied by long-running connections (those that handle over 100 requests).

8.3. The effect of a Web indexer

Analysis of the traces showed that one client was responsible for most of the long runs of HTTP retrievals from the corporate server. This client turned out to be an experimental Web indexing system, which occasionally walked the entire tree of Web pages on the corporate server. Although Web indexers are not uncommon, they do not reflect the usage patterns of latency-sensitive ‘‘real’’ users, so I filtered out this client’s requests from the traces and repeated the simulations.

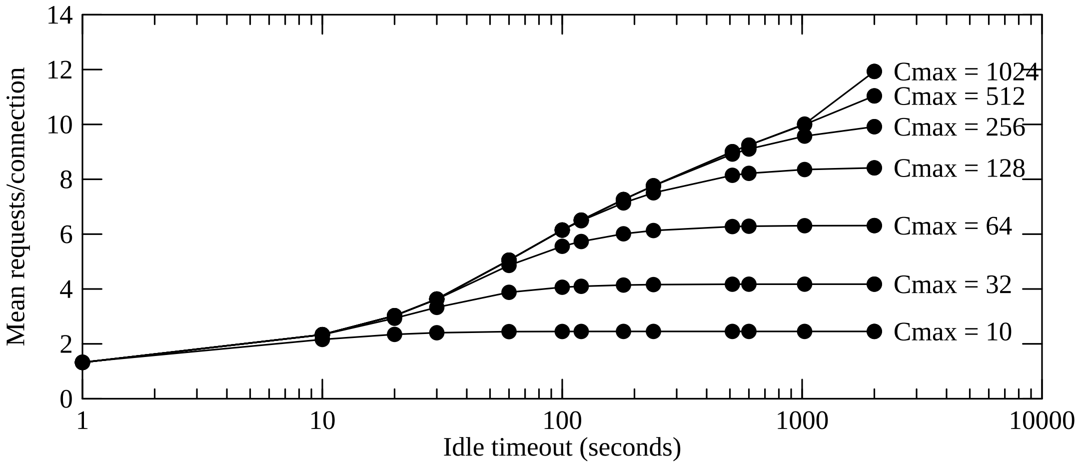
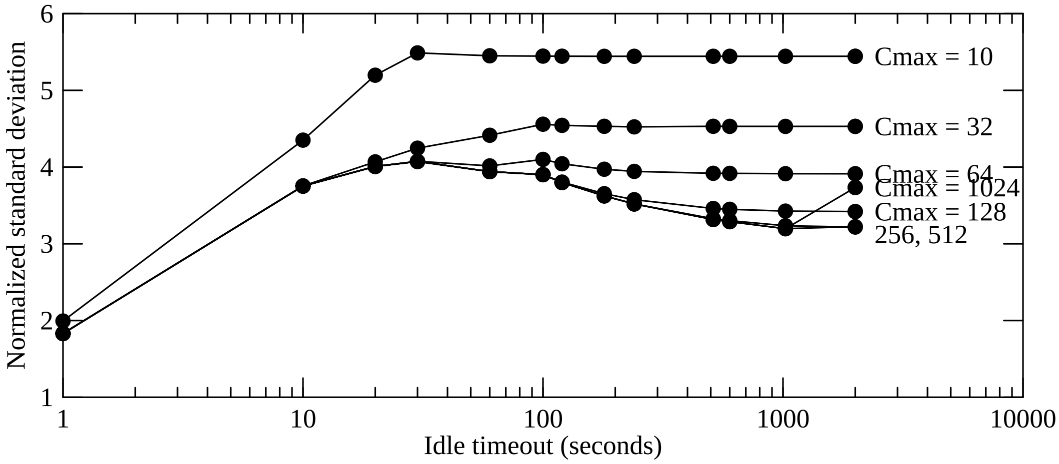


Figure 17: Mean number of HTTP requests per TCP connection (Corporate server)



Standard deviations normalized to means

Figure 18: Standard deviations of requests/connection (Corporate server)

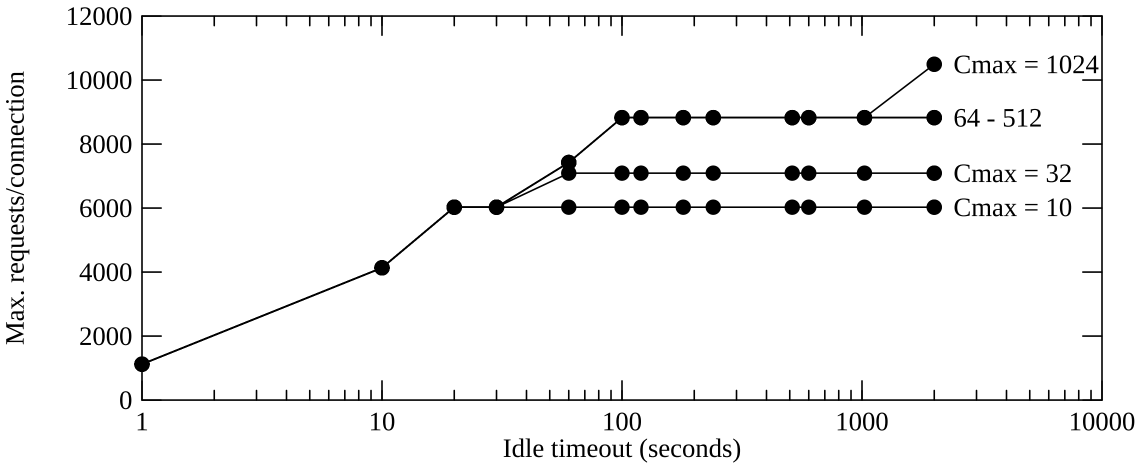


Figure 19: Maximum number of requests/connection (Corporate server)

The means, standard deviations, and maxima of simulations based on the filtered traces are shown in figures 21 through 23. The cumulative distribution of requests per connection is nearly identical to figure 20, except near the right side of the graph, and is not shown here.

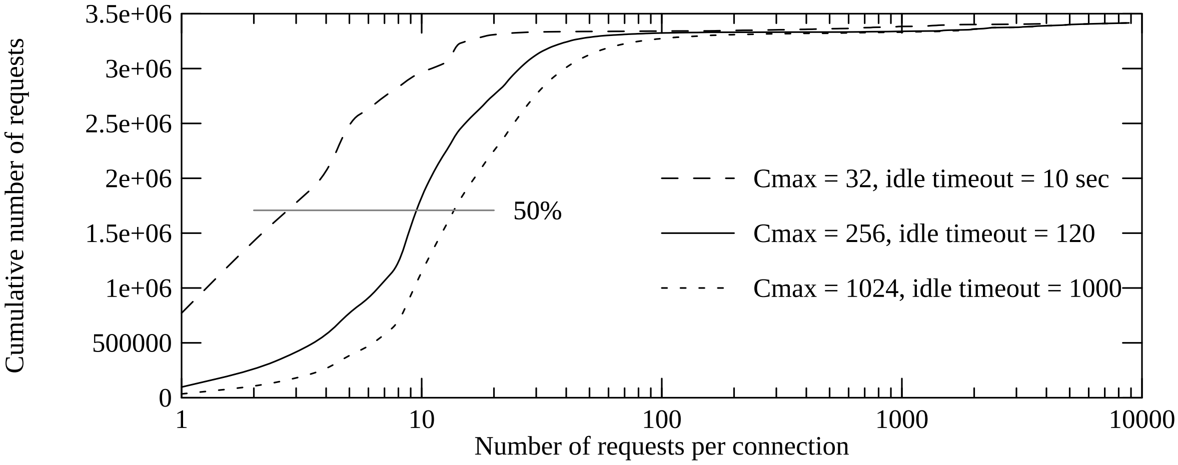


Figure 20: Cumulative number of requests/connection (Corporate server)

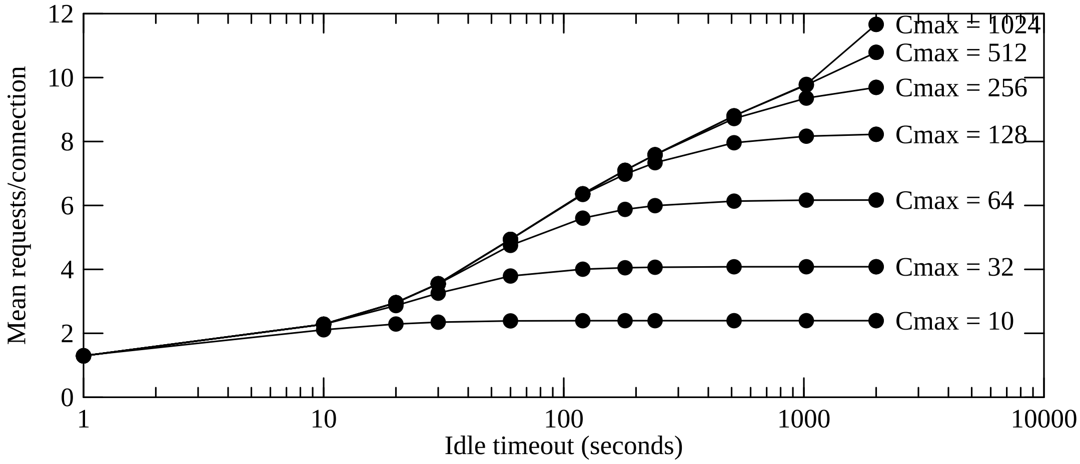
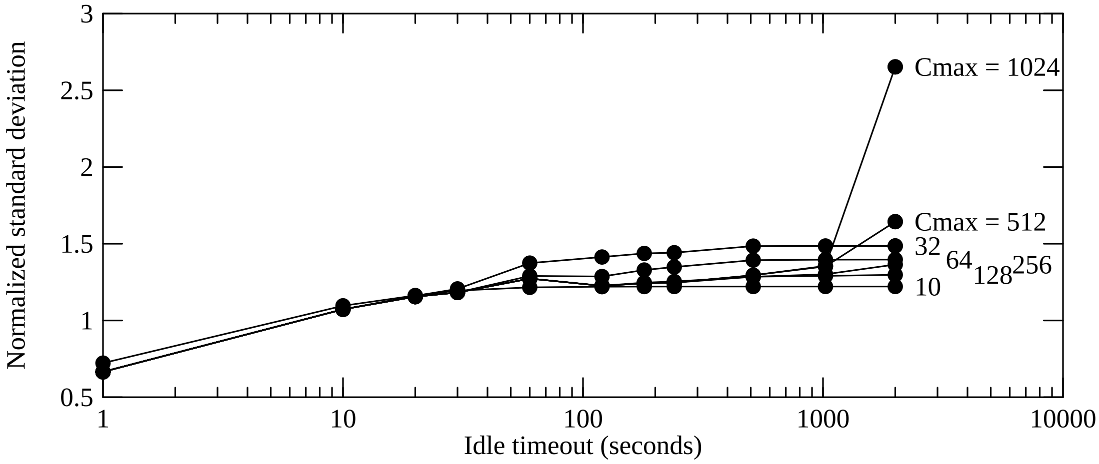


Figure 21: Mean number of requests per connection (Corporate server, no indexer)



Standard deviations normalized to means

Figure 22: Standard deviations of requests/connection (Corporate server, no indexer)

Even with the indexer's activity, which clearly benefits from P-HTTP, removed from the traces, the mean number of requests per connection is only slightly reduced (compare figures 17 and 21). However, the standard deviations and maxima are greatly reduced, with the exception

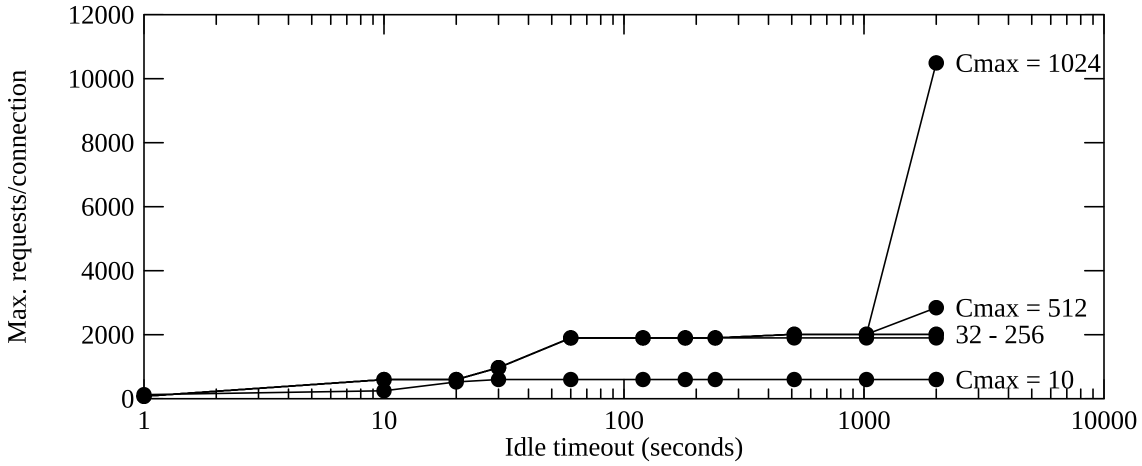


Figure 23: Maximum number of requests/connection (Corporate server, no indexer)

of one additional outlier seen with long idle timeouts. This turned out to be a popular proxy relay.

8.4. Success rates viewed by client

The high maximum values of the requests per connection statistic, as shown in figures 15 and 19, raised the question of what fraction of the client hosts saw a high hit rate. That is, were the TCP connections that saw many HTTP requests limited to a small subset of the client hosts, or were they distributed among a wide set of clients?

Figures 24 and 25 show, for the election service and corporate server respectively, the distribution of the best-case request per connection values among the clients. For example, from figure 24 we can see that, for C_{max} = 256 and an idle timeout of 120 seconds (solid curve), over half of the individual election client hosts would have been able, at least once, to send at least 20 HTTP requests over a single TCP connection.

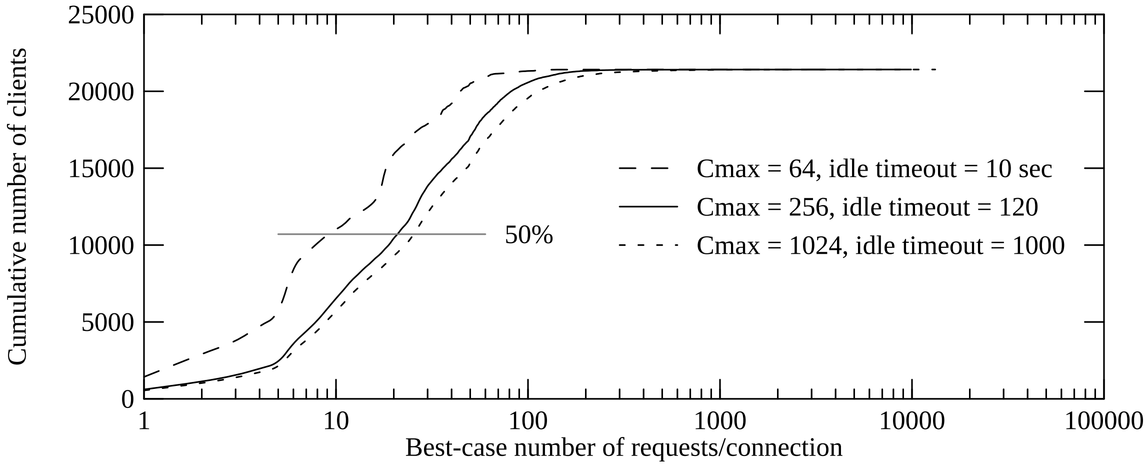


Figure 24: Distribution of best-case requests/connection (Election service)

This wide distribution of the benefits of P-HTTP depends somewhat on the server parameters. For example, figure 25 shows that for the corporate server, setting C_{max} = 32 and the idle timeout to 10 seconds (dashed curve), half of the client hosts would never send more than four HTTP requests over a single TCP connection.

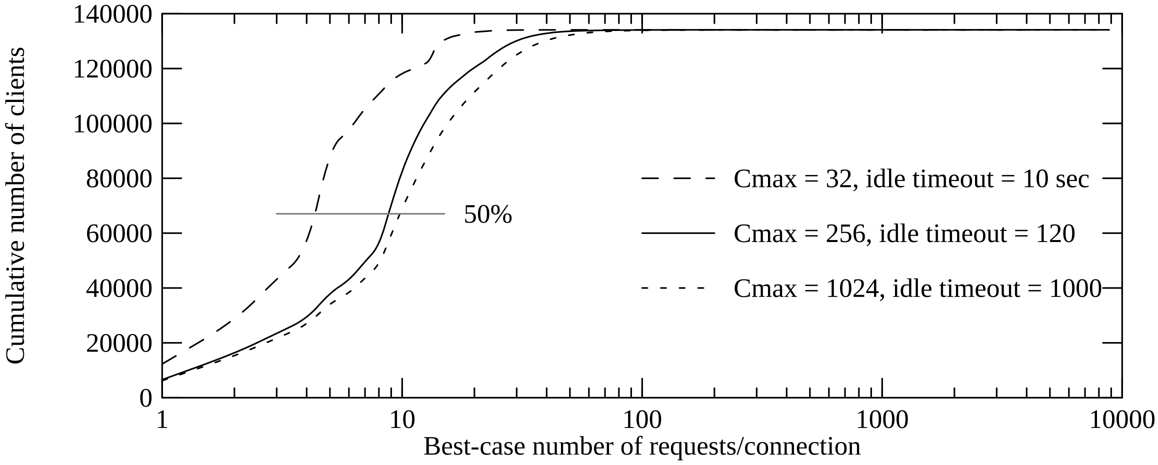


Figure 25: Distribution of best-case requests/connection (Corporate server)

8.5. Frequency of forced closes

A P-HTTP server closes an idle TCP connection for two reasons: either it needs to make room for a request from a different client (a ‘‘forced’’ close), or the connection has been idle for longer than the idle-timeout parameter allows. Figures 26 and 27 show the number of forced closes for various parameter values; the number of connections closed by idle timeouts is the complement of this number.

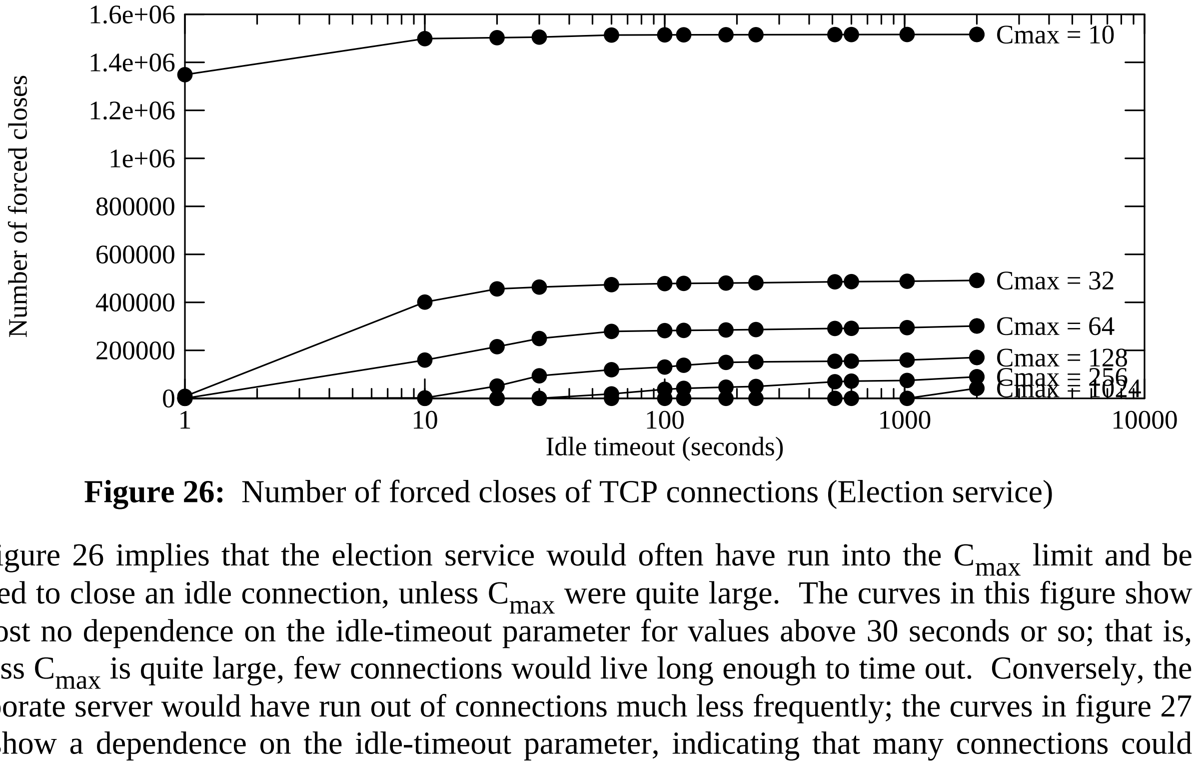


Figure 26: Number of forced closes of TCP connections (Election service)

Figure 26 implies that the election service would often have run into the C_{max} limit and be forced to close an idle connection, unless C_{max} were quite large. The curves in this figure show almost no dependence on the idle-timeout parameter for values above 30 seconds or so; that is, unless C_{max} is quite large, few connections would live long enough to time out. Conversely, the corporate server would have run out of connections much less frequently; the curves in figure 27 do show a dependence on the idle-timeout parameter, indicating that many connections could persist for hundreds or even thousands of seconds.

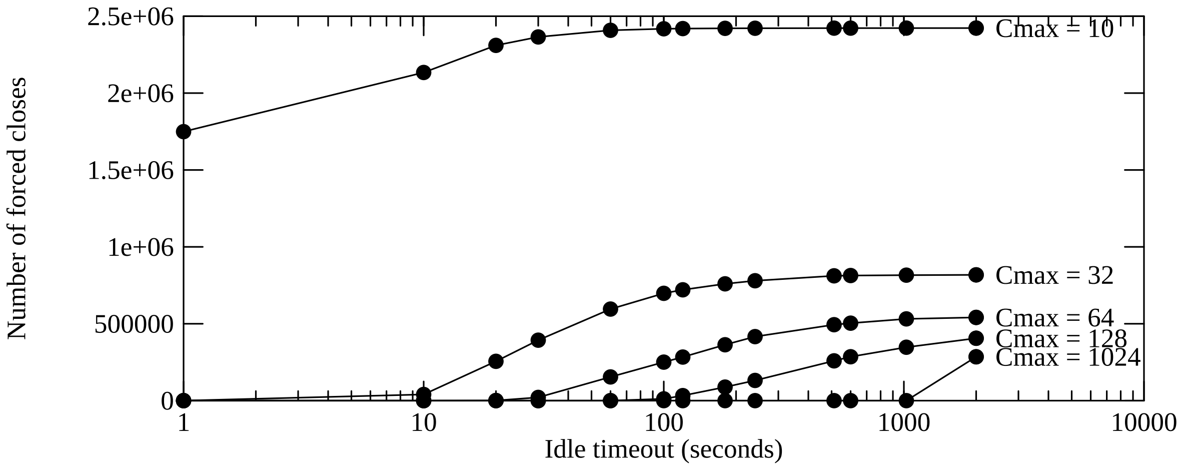


Figure 27: Number of forced closes of TCP connections (Corporate server)

8.6. PCB table use

In section 5.4, I argued that P-HTTP should dramatically reduce the number of PCB table entries in the TIME_WAIT state. Even if the number of PCB table entries for open connections increased somewhat (because the server is holding connections open longer), the total number of PCB table entries should decrease.

The simulator counts the number of ESTABLISHED and TIME_WAIT entries. (The simulator reports the peaks, rather than means, of these counts, because the system must reserve enough memory to satisfy the peak. Also, the peak size determines the CPU-time cost of PCB lookups during periods of heavy load, precisely when this cost is most problematic.) An HTTP server, which closes connections quite rapidly, also ends up with many entries in a variety of short-duration states (primarily CLOSE_WAIT and FIN_WAIT), but the simulator did not model this. A P-HTTP server with a reasonably good open-connection hit rate, and thus a relatively low rate of connection closes, should tie up relatively few PCB table entries in these short-duration states.

Figures 28 and 29 show the peak number of TIME_WAIT entries for various combinations of parameters. P-HTTP always does significantly better than HTTP. This holds true even if one looks at the total number of PCB table entries (modelled as the sum of the TIME_WAIT entries plus C_{max}).

Generally, the number of TIME_WAIT entries does not appear to depend on the idle-timeout parameter, for values above a threshold that varies somewhat with C_{max}. This is because most connections are closed before the idle timeout goes off, and so most of the TIME_WAIT entries are generated by forced closes.

However, for large values of C_{max} (somewhere around 500 connections), increasing the idle-timeout parameter actually increases the number of TIME_WAIT entries. Figures 26 and 27 show that this part of the parameter space is where forced closes are most likely to occur, because the server does not have a big enough pool of free connections to draw on. If connection requests from new hosts, and hence forced closes, occur in bursts, then this would cause a peak in the number of TIME_WAIT entries. Connections closed due to an idle timeout, on the other

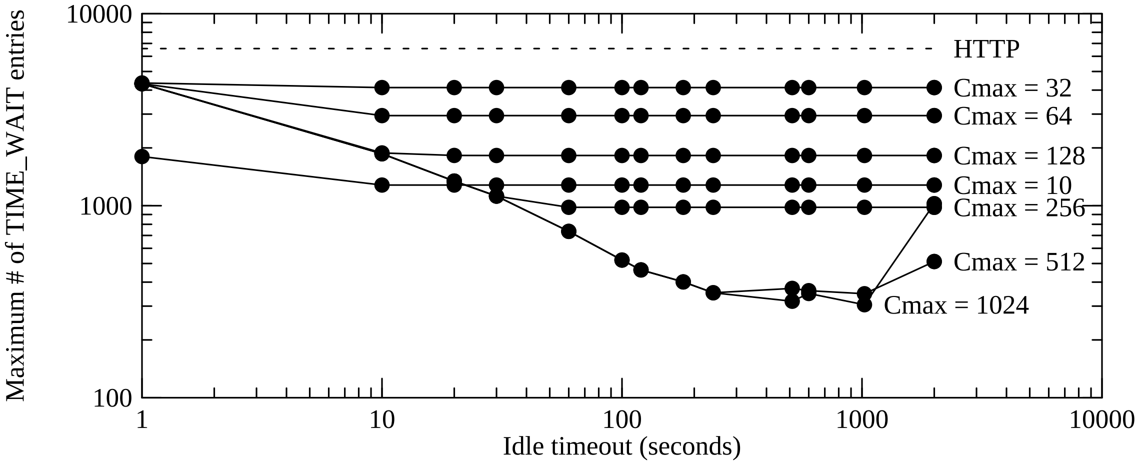


Figure 28: Maximum number of TIME_WAIT entries in PCB table (Election service)

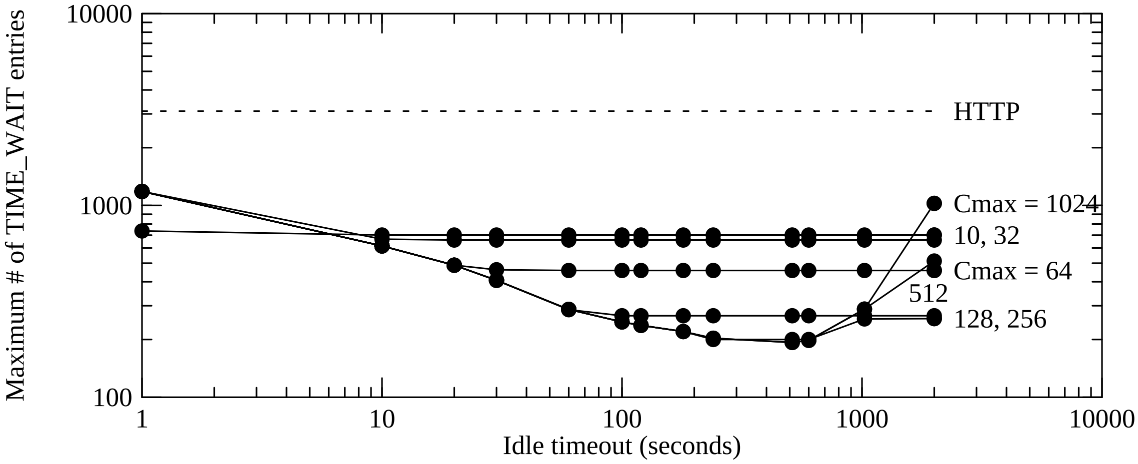


Figure 29: Maximum number of TIME_WAIT entries in PCB table (Corporate server)

hand, may be more spread out in time, and so their TIME_WAIT entries are less likely to coexist in the PCB table.

This effect suggests that increasing the idle timeout without bound, while it might improve the open-connection hit rate slightly, would not be a good idea. Increasing the demand for PCB table entries results in a moderate increase in the memory resource requirements, and potentially a large increase in the CPU overhead for managing this table.

8.7. Adaptive timeouts

How well would P-HTTP perform if servers had to use the adaptive timeout scheme (described in section 5.2) in order to deal with HTTP proxies? That is, what would happen to the open-connection hit rate? A reduction in this rate would reduce the benefit of P-HTTP.

I simulated an adaptive-timeout policy while varying several parameters, including the initial timeout and the size of the per-client timeout table. For these simulations, I held the maximum number of connections constant (C_{max} = 512), in order to avoid confusion.

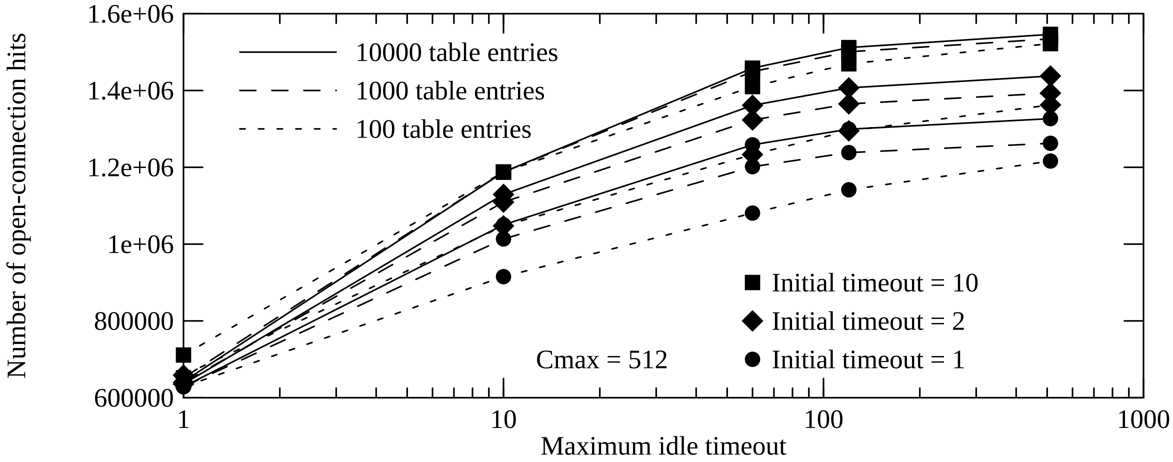


Figure 30: Effect of adaptive timeouts on open-connection hits (Election service)

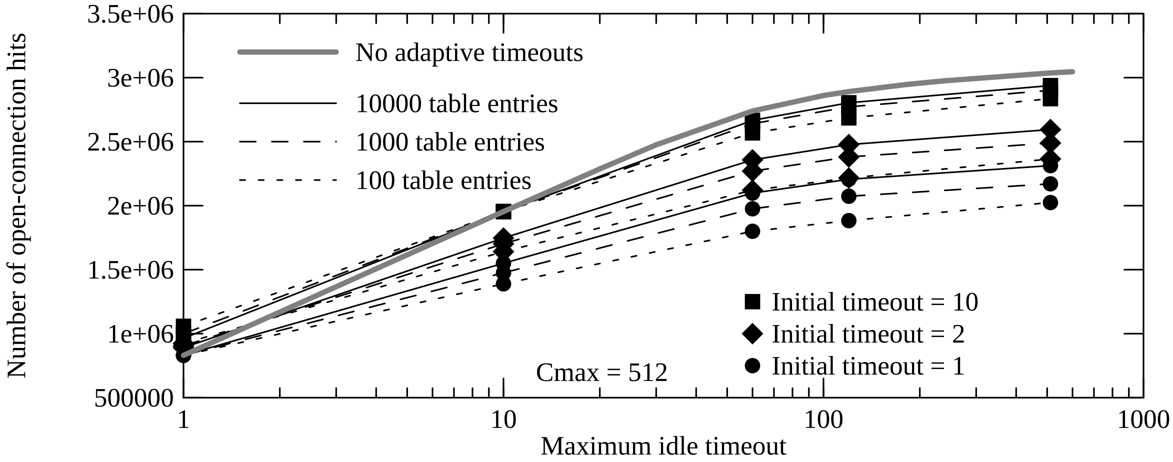


Figure 31: Effect of adaptive timeouts on open-connection hits (Corporate server)

Figures 30 and 31 show the results for the number of open-connection hits. Figure 31 also shows the results for a P-HTTP server without adaptive timeouts (gray curve); figure 30 does not show this curve, which would overlap almost exactly with the curve for the adaptive timeout case with a table size of 10,000 entries, and an initial timeout of 10 seconds.

With the initial timeout set to 1 second, and even with the largest simulated table size, the adaptive timeout scheme fails to capture between 15% (election service) and 24% (corporate server) of the open-connection hits available to the non-adaptive scheme. (Note that the vertical scales in figures 30 and 31 do not start at zero.) So the adaptive timeout scheme does reduce the potential performance of P-HTTP, although it would still eliminate 84% of the election service's TCP connections, and 73% of the corporate server's connections.

8.8. Network loading

How might P-HTTP affect the load on the network itself? It is hard to predict P-HTTP packet arrival patterns based on HTTP traces, but we can make some crude inferences from the simulation results. Under the assumption that the number and size of data packets would not change (although batching of requests and responses could result in some packet-count reduction), the primary statistical effect would be a reduction in the number of TCP ‘‘overhead’’ packets sent. Each TCP connection typically involves seven such packets (for SYNs, FINs, and associated ACKs; see figure 1).

On its busiest calendar day, the election service handled 737,184 HTTP requests, sending and receiving a total of 12.7 million packets (including retransmissions and failed connections). Thus, the average HTTP request involved 17.25 packets. If the server had used P-HTTP with C_{max} = 1024 and a fixed idle timeout of 2 minutes, eliminating more than 95% of the TCP connections, that would have eliminated 4.9 million overhead packets, or 38% of the total packet load.

Use of P-HTTP should also improve the dynamics of the Internet, by reducing the number of times an end-host must discover the TCP congestion-control window size. A 95% reduction in the number of TCP connections should cause a significant improvement in network dynamics ‘‘near’’ the server: that is, in any bottleneck links common to the paths to a large fraction of the clients. However, since we do not know how congested these links really were, it is not possible to estimate how much congestion would have been avoided through the use of P-HTTP.

9. Related work

The argument that P-HTTP will reduce client latency without requiring servers to maintain too many open connections is, at its heart, an assertion that HTTP clients show strong temporal locality of reference. Several other studies have looked at locality of reference in the context of network connections. One [18] showed that processes communicating on a LAN showed moderately high locality of reference, although this study did not specifically look at protocols with short connection durations. Several researchers have looked at the feasibility or performance of intermediary caching servers to reduce the number of high-latency network retrievals of files [8, 19], Web pages [6, 11, 22], and Domain Name Service data [9]. The studies of Web access patterns, in particular, show significant locality of reference.

10. Future work

The simulations presented in this paper only begin to explore the performance implications of P-HTTP. A more advanced simulator could shed additional light. For example, use of P-HTTP should shorten the response time for many requests. This effect will change the order in which requests are received. By making some reasonable assumptions about the distribution of network round-trip times, the simulator could generate a modified event trace reflecting these shorter times. This new semi-synthetic trace could then be fed back to the simulator, producing somewhat more realistic results.

A simulation might also be able to show the effect of P-HTTP on the users’ perception of performance. That is, how rapidly does the client host receive a response to its requests? Again using assumptions about network RTTs, and some semantic information (such as which retrievals are for inlined images), it should be possible to simulate the actual response times seen by the users.

The simulations done for this paper assumed that a server would use LRU algorithms for managing two resources: the pool of TCP idle connections, and the table of recently-active clients (for the adaptive-timeout scheme). What would happen if the server closed the ‘‘least active’’ idle connection (based on a running load average), instead of the least-recently-used connection? Should the server try to keep client hosts that ‘‘hit’’ (i.e., that are obviously using P-HTTP) in the adaptive-timeout table, in preference to more recently active hosts that are not using P-HTTP? And should the server be less eager to time out idle connections if the total number is much less than C_{max}? Simple modifications to the existing simulator could help answer these and similar questions.

The simulation results presented in this paper looked at connection counts over the course of several days or months. How would the relative performance of P-HTTP look over much shorter time scales, such as seconds or minutes? The simulator could be modified to provide periodic statistics (as in figure 7), instead of simply generating totals.

Simulations can only go so far. Ultimately, we will not know how well P-HTTP works until it is widely implemented and heavily used. This experience should also lay to rest any remaining questions about reliability.

A persistent-connection model for Web access potentially provides the opportunity for other improvements to HTTP [20]. For example, if authentication could be done per-connection rather than per-request, that should significantly reduce the cost of robust authentication, and so might speed its acceptance.

11. Summary and conclusions

The simplicity of HTTP has led to its rapid and widespread adoption, and indeed to the explosive growth in the number of Internet users. This simplicity, however, limits the potential performance of HTTP and risks disappointing many of these users. HTTP misuses TCP, so HTTP clients suffer from many unnecessary network round-trips. Meanwhile, HTTP servers must devote excessive resources to TCP connection overheads.

Persistent-connection HTTP can greatly reduce the response time, server overheads, and network overheads of HTTP. The simulations presented in this paper strongly support the intuition that, by exploiting temporal locality of reference, P-HTTP can avoid most of the TCP connections required by HTTP. P-HTTP should also help an overloaded server flow-control its clients.

These simulations also provide some guidance in the choice of P-HTTP server configuration parameters. Generally, increasing the maximum number of active connections (C_{max}) will increase users’ perceived performance, at the expense of somewhat increased server resource demands. Increasing the idle timeout improves UPP and, up to a point, also reduces server PCB table space requirements.

The feasibility of P-HTTP ultimately depends on the availability of robust client and server implementations, and the conversion of proxies. The simulation results presented in this paper, however, suggest that efforts to switch to P-HTTP will be richly rewarded.

Acknowledgements

Venkata Padmanabhan helped design P-HTTP, and implemented a prototype client and server. David Jefferson masterminded the 1994 California election service. Both Venkata and David proofread several drafts of this paper. 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. Stephen Stuart and Richard Schedler helped me obtain logs from the corporate server. I am grateful for the comments of Mike Schwartz and the anonymous SIGCOMM ’95 reviewers.

References

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

[2] Tim Berners-Lee and Daniel W. Connolly. *HyperText Markup Language Specification - 2.0*. Internet Draft draft-ietf-html-spec-00.txt, IETF, November, 1994. This is a working draft.

[3] 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.

[4] R. Braden. *Extending TCP for Transactions -- Concepts*. RFC 1379, University of Southern California Information Sciences Institute, November, 1992.

[5] R. Braden. *T/TCP -- TCP Extensions for Transactions: Functional Specification*. RFC 1644, University of Southern California Information Sciences Institute, July, 1994.

[6] Hans-Werner Braun and Kimberly Claffy. Web traffic characterization: an assessment of the impact of caching documents from NCSA’s web server. In *Proc. Second WWW Conference ’94: Mosaic and the Web*, pages 1007-1027. Chicago, IL, October, 1994.

[7] CompuServe, Incorporated. Graphics Interchange Format Standard. 1987.

[8] Peter B. Danzig, Richard S. Hall, Michael F. Schwartz. A Case for Caching File Objects Inside Internetworks. In *Proc. SIGCOMM ’93 Symposium on Communications Architectures and Protocols*, pages 239-248. San Francisco, CA, September, 1993.

[9] Peter B. Danzig, Katia Obraczka, and Anant Kumar. An Analysis of Wide-Area Name Server Traffic. In *Proc. SIGCOMM ’92 Symposium on Communications Architectures and Protocols*, pages 281-292. Baltimore, MD, August, 1992.

[10] Enterprise Integration Technologies. The Secure HyperText Transfer Protocol. URL http://www.eit.com/projects/s-http/index.html.

[11] 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.

[12] HTTP Working Group of the Internet Engineering Task Force. HTTP-WG Mailing List Archives. URL http://www.ics.uci.edu/pub/ietf/http/hypermail/.

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

[14] Chet Juszczak. Improving the Performance and Correctness of an NFS Server. In *Proc. Winter 1989 USENIX Conference,* pages 53-63. San Diego, February, 1989.

[15] Samuel J. Leffler, Marshall Kirk McCusick, Michael J. Karels, and John S. Quarterman. *The Design and Implementation of the 4.3BSD UNIX Operating System.* Addison-Wesley, Reading, MA, 1989.

[16] 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.

[17] Robert B. Miller. Response Time in Man-Computer Conversational Transactions. In *Proc. American Federation of Information Processing Societies Conference,* pages 267-277. Fall, 1968. Vol. 33 pt. 1.

[18] Jeffrey C. Mogul. Network Locality at the Scale of Processes. *TOCS* 10(2):81-109, May, 1992.

[19] D. Muntz and P. Honeyman. Multi-level Caching in Distributed File Systems -or- Your cache ain't nuthin' but trash. In *Proc. Winter 1992 USENIX Conference*, pages 305-313. San Francisco, CA, January, 1992.

[20] Venkata N. Padmanabhan. Improving World Wide Web Latency. Master's thesis, University of California at Berkeley, May, 1995.

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

[22] James E. Pitkow and Margaret M. Recker. A Simple yet Robust Caching Algorithm Based on Dynamic Access Patterns. In *Proc. Second WWW Conference '94: Mosaic and the Web*, pages 1039-1046. Chicago, IL, October, 1994.

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

[24] Dave Raggett. *HyperText Markup Language Specification Version 3.0*. Internet Draft draft-ietf-html-specv3-00.txt, IETF, March, 1995. This is a work in progress.

[25] Richard Rubenstein and Harry M. Hersh with Henry F. Ledgard. *The Human Factor: Designing Computer Systems for People*. Digital Press, Burlington, MA, 1984.

[26] Russel Sandberg, David Goldberg, Steve Kleiman, Dan Walsh, and Bob Lyon. Design and Implementation of the Sun Network filesystem. In *Proc. Summer 1985 USENIX Conference*, pages 119-130. Portland, OR, June, 1985.

[27] Simon E. Spero. Analysis of HTTP Performance problems. URL http://elanor.oit.unc.edu/http-prob.html. July, 1994.

[28] Simon E. Spero. Message to IETF HTTP working group. Message ID <9412162131.AA16636@tipper.oit.unc.edu>. December, 1994.

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.

WRL Technical Notes

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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