Web Proxy Caching: The Devil is in the Details

Ramón Cáceres Fred Douglis Anja Feldmann

Gideon Glass* Michael Rabinovich

AT&T Labs–Research

Florham Park, NJ, USA

{ramon,douglis,anja,misha}@research.att.com, gid@cobaltmicro.com

Abstract

Much work in the analysis of proxy caching has focused on high-level metrics such as hit rates, and has approximated actual reference patterns by ignoring exceptional cases such as connection aborts. Several of these low-level details have a strong impact on performance, particularly in heterogeneous bandwidth environments such as modem pools connected to faster networks. Trace-driven simulation of the modem pool of a large ISP suggests that "cookies" dramatically affect the cacheability of resources; wasted bandwidth due to aborted connections can more than offset the savings from cached documents; and using a proxy to keep from repeatedly opening new TCP connections can reduce latency more than simply caching data.

1 Introduction

The continued growth of the World Wide Web (WWW) motivates techniques to improve its performance. One popular technique is *proxy caching*, in which one or more computers act as a cache of documents for a set of WWW clients. These clients are configured to send HyperText Transport Protocol (HTTP) requests to the proxy. If possible, the proxy serves the requests from its cache. Otherwise, the proxy forwards the request to the *content provider*, that is, to the server containing the source copy of the requested data.

WWW proxy caching attempts to improve performance in three ways. First, caching attempts to reduce the user-perceived latency associated with obtaining Web documents. Latency can be reduced because the proxy cache is typically much closer to the client than the content provider. Second, caching attempts to lower the network traffic from the Web servers. Network load can be lowered because documents that are served from the cache typically traverse less of the network than when they are served by the content provider. Finally, proxy caching can reduce the service demands on content providers since cache hits need not involve the content provider. It also may lower transit costs for access providers.

Many attempts to model the benefits of proxy caching have looked only at the high-level details. For instance, one might consider a stream of HTTP request-response pairs, knowing the URL requested and the size of each response, and compute a *byte hit ratio*: the fraction of the number of bytes served by the cache divided by the total number of bytes sent to its clients. Hit rates estimate the reduction in bandwidth requirements between the proxy and the content providers; however, they do not necessarily accurately reflect the latency perceived by the end user, and in some cases they do not even reflect actual bandwidth demands.

Kroeger et al. examined the effect of caching and prefetching on end-user latency [6]. They found that proxy caching

*Current affiliation: Cobalt Microserver, Inc., Mountain View, CA, USA

could reduce latency by 26% at most, which was much less than the 77% or more of latency that was attributed to communication between the proxy and the content providers. Already, by looking at latency in addition to bandwidth, they demonstrated some of the limitations of proxy caches. However, they made some other simplifying assumptions, such as ignoring requests that did not result in a successful retrieval. We have experimental evidence that suggests that aborted transfers can contribute significantly to total bandwidth requirements, and the use of a proxy actually can increase traffic (by as much as 18% in our study) because of the greater amount of data in transit at the time a client aborts a request.

As another example of the importance of detail, consider the effect of *cookies*. Gribble and Brewer studied a large client population and found that the hit ratio for a proxy cache could approach 60% [4]. However, despite examining a number of factors that might contribute to particular resources *not* being cacheable, they omitted a significant contributing factor, cookies. Since cookies are methods of customizing resources on a per-user basis, it is inappropriate to cache HTTP/1.0[1] resources that have cookies in them. In the client trace described here, we found roughly 30% of requests had cookies, which dramatically limits the number of requests that can possibly be satisfied from a proxy cache.

Thus, we argue that to judge the performance impact of proxy caches, one needs to take the interactions between HTTP, TCP, and the network environment into consideration. In addition, one needs to consider all possible factors that affect individual transfers, such as restrictions on cacheability or data in transit during connection aborts.

We have developed a web proxy cache simulator that attempts to provide this degree of realism. It uses a workload from a trace that was collected in a production environment, AT&T Worldnet. It simulates HTTP operations at the level of individual TCP packets, including the slow-start phase at the beginning of each connection. This simulation includes data in transit, demonstrating the effects of connection aborts. It uses all packet request and response headers, so information that affects cacheability, such as the presence of cookies, is included. Finally, it uses a combination of measured and parameterizable timing characteristics, allowing us to simulate a user community over a relatively slow modem pool, or a faster local area network. In simulating the timing characteristics for the modem pool, we found that the benefits of using persistent connections between clients and the proxy cache can outweigh the benefits of caching documents.

[1] In HTTP/1.1, caching and cookies are decoupled, and a server is responsible for explicitly disabling caching when appropriate.

2 Tracing Environment

We gathered traces from a FDDI ring that connects an AT&T Worldnet modem bank to the rest of the Internet. The modem bank contains roughly 450 modems connected to two terminal servers, and is shared by approximately 18,000 dialup users. The servers terminate Point to Point Protocol (PPP) connections and route Internet Protocol (IP) traffic between the users and the rest of the Internet.

We collected raw packet traces on a dedicated 500-MHz Alpha workstation attached to the FDDI ring. We insured that the monitoring was purely passive by configuring the workstation's FDDI controller so that it could receive but not send packets, and by not assigning an IP address to the FDDI interface. We controlled the workstation by connecting to it over an internal network that does not carry user traffic. We made the traces anonymous by encrypting IP addresses as soon as they came off the FDDI link, before writing any packet headers to stable storage.

We traced all dialup traffic on the FDDI ring for 12 days in mid-August, 1997. During this time, 17,964 different users initiated 154,260 different dialup sessions, with a maximum of 421 simultaneously active sessions. Our tracing instrument handled more than 150 million packets a day with less than 0.3%

packet loss.

We processed the raw packet stream on the fly to obtain our final trace. The resulting trace contains all relevant information and is much more compact than the raw packet data. The trace records the following information for TCP conversations in which one port number is the default HTTP port (80):

- TCP events: Timestamps, sequence numbers, and acknowledgments for all packets with SYN, FIN, or RST bits set.
- HTTP events: Timestamps for HTTP requests from clients and HTTP responses from servers, and timestamps for the packets containing the first and the last portion of the data in each direction.
- HTTP headers: Complete HTTP headers for both requests and responses.
- Byte counts: Count of bytes sent in either direction for HTTP header and body, if present.

This information is sufficient to determine how much time is taken by various components of the observed HTTP conversations. These traces are significantly more detailed than traces that other researchers have made available [4, 6]. For example, publicly-available traces lack timestamps for TCP events such as connection creation (SYN packets).

3 Simulator

Our web proxy simulator, named *PROXIM*, simulates a proxy cache using the HTTP trace (described in Section 2) as input. *PROXIM* can be used to simulate the three different scenarios shown in Figure 1,

a. using a proxy,

b. without a proxy, where the bandwidth to the clients is the bottleneck, and

c. without a proxy, where the bandwidth on the network connecting the clients to the Internet is the bottleneck.

To assess the performance impact of proxy caching in a given environment, we compare the with-proxy to the without-proxy simulation results. In addition, the without-proxy simulation results are used to compare the simulation results against the original measured numbers.

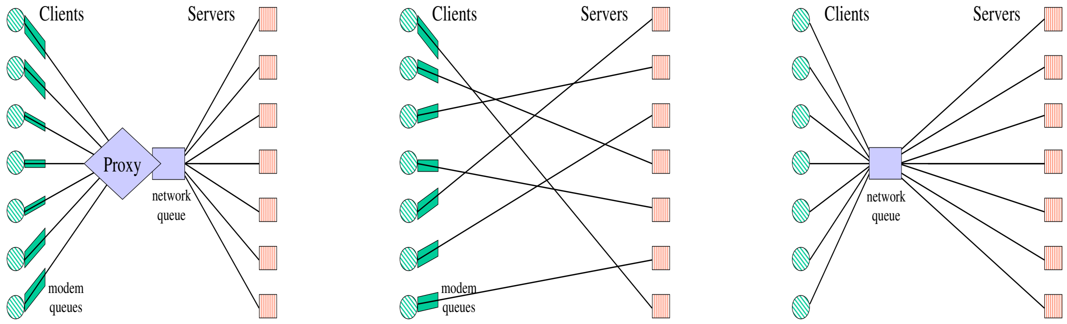
The following subsections describe various aspects of *PROXIM*'s functionality.

3.1 Simulated Cache

PROXIM simulates a document cache managed by LRU replacement. However in all of our experiments we configured the cache size to be sufficiently large to avoid having to evict any documents. This gives potentially optimistic results for the proxy, but the storage consumed by all cacheable documents in our trace is on the order of 40GB, not an unreasonable amount of disk space. To account for the overhead introduced by the proxy a configurable cache miss time overhead and a configurable cache hit time overhead are added to the request service time (during cache hits and cache misses, respectively).

3.2 Network Connections

As we argue in the introduction, it is important to handle the interactions between HTTP and TCP, particularly persistent HTTP [3, 7, 9]. In *PROXIM*, each client maintains zero or more open connections to the proxy. An *idle* connection is one in which no request is currently active. When a persistent HTTP request is generated and an idle connection exists for the client in question, an idle connection is chosen to service the request. By default, requests that were marked non-persistent in the original trace are treated as non-persistent by *PROXIM* and result in a new connection. If



(a) Using a proxy.

(b) No proxy; client bandwidth is the bottleneck.

(c) No proxy; bandwidth to the Internet is the bottleneck.

Figure 1: *Proxy caching environment.*

a client generates a request and all of its extant proxy connections are active servicing other requests—or if the client has no extant connections—a new TCP connection is created by the client to service the present request. Client-to-proxy connections that are idle for more than a default of 3 minutes are closed by the proxy.

PROXIM handles proxy to content-provider connections similarly to the way it handles proxy to client connections. However, since proxy administrators have no influence over arbitrary content-providers, we time out persistent server connections after a somewhat arbitrary default of 30 seconds of idle time.

PROXIM assumes that a client requests a Web page at the time that the original client sent out the SYN packet (TCP connection request). Therefore the proxy learns about an HTTP request at the time at which we recorded the SYN packet if the client can reuse a persistent connection, or otherwise one round-trip time after the original SYN packet was observed.

3.3 Document Transfer

PROXIM simulates the transfer of documents to clients over a connection by scheduling individual packets to arrive at the client, the proxy, or the server. The scheduling of packets takes into account TCP slow-start at the start of a connection, but does not consider packet losses and the additional slow-start periods those would entail. By default, the proxy and servers send 1500-byte packets (the most common packet size besides 40-byte ACK packets).

Packets destined for the clients are enqueued in a per-client queue for scenarios (a) and (c) and in a network queue for scenario (b) of Figure 1. Packets destined for the proxy are enqueued in a network queue. Each of these queues drains at a configurable rate. The default for the server-to-proxy rate is 45 Mbps while the default rate for client queues is 21 Kbps. The first was chosen since a T3 constitutes a good connectivity to the Internet and the later was chosen since most modems connection speeds are in the order of 24 Kbps to 28 Kbps and we observed that they can sustain a throughput around 21 Kbps.

This scheduling of packets is only possible given a round-trip time estimate. We use a constant round-trip time estimate for each connection, which for client-to-proxy connections is the modem delay (default 250 ms). For proxy-to-server connections the round-trip time is derived from either the difference between the SYN, SYN-ACK timestamps or the REQUEST, RESPONSE timestamps. (We are assuming that the proxy is located at the location of the packet sniffer, where the trace was collected.) The estimate for client-to-server connection is done in a similar manner except that we add the modem delay of 250 ms.

3.4 Latency Calculations

PROXIM simulates the overall latency to retrieve a document by breaking the latency overhead into connection setup time, HTTP request-response overhead, and document transfer time. Connection setup costs are avoided when a persistent connection is reused. In the without-proxy case (in which we simulate direct connections between clients and content-providers) we use the SYN timestamps in the trace. In the with-proxy case, when a client establishes a connection with the proxy, or the proxy does so with a content-provider, we use the relevant subset of the SYN packet timestamps. Similarly, request-response latency is calculated using the appropriate portion of the HTTP request/reply latencies in the trace: the full latency is used if the proxy has to contact the content provider; otherwise the request/response latency is assumed to be the configured client/proxy round-trip time. We modeled the time to retrieve data from the proxy by using the queues mentioned above, which are draining at modem bandwidth. We break documents into packets and observe the time difference between the start of the first downloaded packet and the queuing of the last packet.

4 Performance Effects of the Proxy

This section describes some insights into performance effects of a proxy within an ISP, gained by considering low-level details that are often overlooked in similar studies. The main lesson is that considering these details has a significant effect on conclusions in all aspects of system performance: hit ratios, bandwidth savings, and user-perceived latency.

4.1 Hit Ratio

Most of performance studies of proxies to date concentrated on hit ratios. We contend this is only a secondary measure, because the correlation between hit ratio and other important metrics, such as bandwidth savings and latency reduction, is low.

Still, even when looking at just hit ratio, considering low-level details provides some interesting results. Unlike existing studies, our trace captures all HTTP headers of requests and responses, so we were able to emulate the full behavior of proxies with respect to caching. In particular, our simulated proxy does not satisfy from the cache any request that comes with a cookie. Surprisingly, the trace showed that over 30% of all requests had a cookie, which had a dramatic effect on the hit ratio. If the proxy ignored cookies, the hit ratio would have been 54.5%, in line with previous studies [4, 6]. By taking cookies into account, we observe the hit ratio of just 35.2%. The byte hit ratio similarly decreases from 40.9% to 30.42%.

The significant increase of hit ratios produced by caching documents with cookies supports the importance of techniques aimed at enabling caching of such of documents. These techniques include approaches based on delta encoding [1, 5, 8] and client-side HTML macro-preprocessing [2]. The extent of the benefits of these techniques would depend on the amount of document customization based on the request cookies.

4.2 Bandwidth Savings

An often-cited benefit of proxy caching is reduced network traffic on the Internet. Surprisingly, by carefully modeling the system behavior when requests are aborted, we found that the proxy can actually *increase* the traffic from content providers to the ISP. Without a proxy, when a client aborts a request, the transfer of the document is interrupted and no longer consumes network bandwidth. On the other hand, due to the bandwidth mismatch between the connections from client to proxy and from proxy to the Internet, the proxy may download much of the document by the time it learns of the abort. Thus the bandwidth consumption of aborted requests is higher when the proxy is present.

The exact effect of this phenomenon on the overall bandwidth demands depends on the handling of aborted requests by the proxy. If the proxy continues the download, subsequent requests to the document will hit in the cache, improving performance. On the other hand, this may further increase the traffic from the content providers to the proxy.

In our experiments, the total number of bytes received by the clients from the content providers without the proxy was 49.8 GB. At one extreme, if the proxy always continued to download upon client's abort, the total number of bytes from the content providers to the proxy would be 58.7 GB (118% of the original number of bytes transferred from the content providers). This means that, with the proxy, one could end up consuming 18% more bandwidth to the Internet.

On the other extreme, the proxy could abort the download immediately after receiving a client's abort. In this case, the the amount of wasted data transmitted depends on the proxy-to-server bandwidth. For the network queue bandwidth of 45 Mbps, aborting downloads would reduce wasted data transmissions by only 5.0 GB, still an increase of 8% over the without-proxy case. A 1.5 Mbps Internet connection would save 8.0 GB of wasted data (corresponding to a 2% increase in overall traffic). However, if the bandwidth of the network queue were just 0.5 Mbps[2], savings from caching data would finally offset any additional transfers after abort, though the savings would be just 6% overall (11.8 GB less unneeded transfers relative to the case where downloads continue after aborts). These bandwidth savings from the proxy are nowhere near those suggested by the byte hit ratio, if they exist at all.

Using traces from clients connected to the Internet via a T1 link, we confirmed that the bandwidth mismatch between the modems and the Internet contributes significantly to the wasted bandwidth consumption from aborts. When a well-connected client aborts a transfer, it will have received a significant fraction of the data received by the proxy. Handling of aborts is important even in the case of well-connected clients, however, since continuing to download to the proxy after an abort incurs about 14% overhead, roughly equivalent to the savings from caching in the first place.

[2]Note that the lower-bandwidth network queues are appropriate models given that a lot of web sites are accessed over a network path that includes at least one T1 or lower speed connection.

4.3 Latency Reduction: Caching Connections vs. Caching Data

One would expect that those requests that are served from the cache will be serviced faster. However, by modeling details of network transfer, including TCP connection setup and packet-level delivery with TCP slow start, we found that caching reduces the mean latency experienced by the user by 3.4%, and the median by 4.2%. This result is substantially more pessimistic than the upper bound of 26% latency reduction reported by Kroeger et al. [6]. The limited improvement is due to the high latency of the connection set-up (which had a mean of 1.3s in the trace), the effect of cookies described earlier, and modem bandwidth limitation on the clients (this study assumed 28.8 Kbps modems).

Due to the high cost of TCP connection setup, potentially significant benefits could be obtained by maintaining persistent connections between clients and servers. However, content-providers can maintain only a limited number of such connections. This suggests a non-traditional role of the proxy as a *connection cache*, which would maintain persistent connection with content providers and re-use them for obtaining documents for multiple clients. The connection cache requires only one (or a small number of) persistent connection(s) to a given content provider, regardless of the number of clients connected to the proxy. Similarly, each client needs to maintain only a small number of persistent connections to the proxy regardless of the number of servers it visits. Moreover, if for some reason a connection between the proxy and client or between the proxy and content provider is torn down, significant performance benefits could still be obtained due to the connection with the other side.

When the proxy is allowed to act as a connection cache, the total mean latency improvements grow to 24% assuming all connections are persistent. Not all Web clients, proxies, and servers may allow persistent connections. If we consider persistent connections only for those requests that specifically allowed persistent connections in the request headers the total mean latency improvement is 13%. The rate of re-use of connections (the hit ratio in the connection cache) from the client to the proxy was found to be up to 81.5% and from the proxy to the server up to 79.5%. These numbers were observed using a 30-second timeout for persistent connections between the proxy and content-providers. We also found that the proxy had to maintain at most 238 simultaneous connections to the servers during these experiments, an easily achievable number.

It is clear that much of the latency reduction that the proxies can achieve is due to cached connections. Thus, an important issue to study, and one that has been largely ignored in the past work, is the policy for managing a connection cache by the proxy. This includes such questions as when to tear down a connection, which connections to tear down when the proxy runs out of connections, how many simultaneous connections it should maintain with the same content provider or the same client, etc.

5 Summary

In this paper, we briefly described some lessons learned from a low-level simulation of a proxy cache within an ISP with dialup users. Our main observation is that considering low-level details has a significant effect on all aspects of system performance: hit ratios, bandwidth savings, and user-perceived latency. For dialup users we found hit ratios to be lower than those reported previously, bandwidth savings non-existent or negative, and latency reduction coming mostly from caching TCP connections rather than documents.

6 Acknowledgments

We would like to thank Albert Greenberg and John Friedmann for their help with the data collection effort. Thanks also to Peter Danzig and Jennifer Rexford for helpful discussions.

References

[1] Gaurav Banga, Fred Douglis, and Michael Rabinovich. Optimistic deltas for WWW latency reduction. In *Proceedings of 1997 USENIX Technical Conference*, pages 289–303, Anaheim, CA, January 1997.

[2] Fred Douglis, Antonio Haro, and Michael Rabinovich. HPP: HTML macro-preprocessing to support dynamic document caching. In *Proceedings of the Symposium on Internetworking Systems and Technologies*, pages 83–94. USENIX, December 1997.

[3] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee, et al. RFC 2068: Hypertext transfer protocol — HTTP/1.1, January 1997.

[4] Steven D. Gribble and Eric A. Brewer. System design issues for internet middleware services: Deductions from a large client trace. In *Proceedings of the Symposium on Internetworking Systems and Technologies*, pages 207–218. USENIX, December 1997.

[5] Barron C. Housel and David B. Lindquist. WebExpress: A system for optimizing Web browsing in a wireless environment. In *Proceedings of the Second Annual International Conference on Mobile Computing and Networking*, pages 108–116, Rye, New York, November 1996. ACM.

[6] Thomas M. Kroeger, Darrell D. E. Long, and Jeffrey C. Mogul. Exploring the bounds of web latency reduction from caching and prefetching. In *Proceedings of the Symposium on Internetworking Systems and Technologies*, pages 13–22. USENIX, December 1997.

[7] Jeffrey Mogul. The case for persistent-connection HTTP. In *Proceedings of ACM SIGCOMM'95 Conference*, pages 299–313, October 1995.

[8] Jeffrey Mogul, Fred Douglis, Anja Feldmann, and Balachander Krishnamurthy. Potential benefits of delta-encoding and data compression for HTTP. In *Proceedings of ACM SIGCOMM'97 Conference*, pages 181–194, September 1997.

[9] Henrik Frystyk Nielsen, James Gettys, Anselm Baird-Smith, Eric Prud'hommeaux, Hakon Wium Lie, and Chris Lilley. Network performance effects of HTTP/1.1, CSS1, and PNG. In *Proceedings of ACM SIGCOMM'97 Conference*, pages 155–166, September 1997.