Proceedings of the Fourth International Web Caching Workshop, San Diego, California
April, 1999, pages 170-178

Simultaneous Proxy Evaluation

Brian D. Davison

Department of Computer Science
Rutgers, The State University of New Jersey
New Brunswick, NJ 08903 USA

davison@cs.rutgers.edu
http://www.cs.rutgers.edu/~davison/

Abstract

The Simultaneous Proxy Evaluation (SPE) architecture is designed to evaluate multiple web proxies in parallel using object requests which are duplicated and passed to each proxy. The SPE architecture reduces problems of unrealistic test environments, dated and/or inappropriate workloads, and is additionally applicable to content-based prefetching proxies. It is intended to measure byte and object hit rates, client-perceived latencies, and cache consistency. We characterize a space of proxy evaluation methodologies and place this architecture within it.

1 Introduction

This paper presents a new architecture for the evaluation of proxy caches. Initially, it grew out of research in techniques for prefetching in web caches. In particular, we found that existing mechanisms for the evaluation of proxy caches were not well suited to prefetching systems. Objective evaluation is paramount to all research, whether applied or academic. Since this is certainly relevant when exploring various approaches to prefetching, we considered the range of existing proxy evaluation methods, and found that they each had distinct disadvantages. This led to the design of the more general architecture described below, and to the ongoing development of a proof-of-concept prototype which will be outlined further at the end of this paper.

Our evaluation method, the Simultaneous Proxy Evaluation (SPE) architecture, reduces problems of unrealistic test environments, dated and/or inappropriate workloads and allows for the evaluation of a larger class of proxies. In the next section we provide a quick introduction to caching on the web, which is followed by a section in which we motivate the need for the SPE architecture by describing the space of proxy evaluation methodologies. We then specify our new evaluation architecture, and comment on implementation issues. We conclude the paper by describing a set of sample evaluation tasks that utilize it and report on the status of our prototype implementation.

2 Background

Caching has a long history and is a well-studied topic in the design of computer memory systems, in virtual memory management in operating systems, in file systems, and in databases. Caching on the Internet is also performed for network services such as DNS and ARP.

Web caching is the temporary storage of web objects (such as HTML documents) for later retrieval.[1] Proponents of web caching claim three significant advantages to web caching: reduced bandwidth consumption (fewer requests and responses that need to go over the network), reduced server load (fewer requests for a server to handle), and reduced latency (since cached responses are available immediately, and closer to the client being served). A fourth is sometimes added: more reliability, as some objects may be retrievable via cache even when the original servers are not reachable. Together, these features

[1] Actually, web caches store HTTP responses, but we will use the more generic phrase, web object, as a simplification throughout.

can make the World Wide Web less expensive and better performing. One drawback of caching is the potential of using an out-of-date object stored in a cache instead of fetching the current object from the server, but this can be minimized by appropriate cache management. Another is the lack of logs of client viewings of pages for the purposes of advertising (although this is being addressed, e.g. [ML97]).

Caching can be performed by the client application, and is built into most browsers. Caching can also be utilized between the client and the server as part of a proxy. *Proxy caches* are often located near network gateways to reduce the bandwidth required over expensive dedicated Internet connections. When shared with other users, these systems can serve many clients with cached objects from many servers. In fact, much of the usefulness (up to 85% of the in-cache requests [DMF97]) is in caching objects requested by one client for later retrieval by another client. For even greater performance, many proxy caches are part of cache hierarchies, in which a cache can request documents from neighboring caches instead of fetching them directly from the server. Finally, caches can be placed directly in front of a particular server, as an *httpd accelerator* to reduce the number of requests the server must handle [CDN^{+}96, Wes98].

A shared proxy cache serves a population of clients. When a proxy cache receives requests for a web object, it checks to see if the response is available in memory or disk, and if so, returns the object to the client without disturbing the upstream network connection or destination server. If it is not available in the cache, the proxy attempts to fetch the object directly.

Proxy caches are increasingly used around the world to reduce bandwidth and alleviate delays associated with the world wide web. This paper describes a method to simultaneously evaluate competing proxy caches on the same request stream. It objectively measures proxy latency, hit rates, and staleness by forming a wrapper around the tested proxies to monitor all input and output.

3 Evaluating Proxy Cache Performance

It is useful to be able to assess the performance of proxy caches, both for consumers selecting the appropriate system for a particular situation and also for developers working on alternative proxy mechanisms. By evaluating performance across all measures of interest, it is possible to recognize drawbacks with particular implementations. For example, one can imagine a proxy with a very large disk cache that provides a high hit rate, but because of poor disk management it noticeably increases the client-perceived latency. One can also imagine a proxy that achieves a high hit rate by caching images only for a very short time which frees storage for use by other objects and just prefetching the inline images for the page currently requested. This would allow it to report high request hit rates, but not perform well in terms of bandwidth savings. Finally, it is possible to select a proxy based on its performance on a workload of requests generated by dialup users and have it perform unsatisfactorily as a parent proxy for a large corporation with other proxies as clients. These examples illustrate the importance of appropriate evaluation mechanisms for proxy systems.

3.1 The Space of Cache Evaluation Methods

The most commonly used cache evaluation method is that of simulation on a benchmark log of object requests. The byte and page hit rate savings can then be calculated as well as estimates of latency improvements. In a few cases, an artificial dataset with the necessary characteristics (appropriate average and median object sizes, or similar long-tailed distributions of object sizes and object repetitions) is used. More commonly, actual client request logs are used since they arguably better represent likely request patterns and include exact information about object sizes.

We characterize web/proxy evaluation architectures along two dimensions: the source of the workload used and form of algorithm implementation. Table 1 shows three categories of data sources for testing purposes, and three forms of algorithm evaluation. For the systems represented in each area of the space, we find it useful to consider how the desired qualities of web caching (reduced bandwidth/server load and improved response time) are measured. In the rest of this section we point out the characteristic qualities of systems in each area of the space. In other work [Dav99], we additionally survey web evaluation research and categorize them by the evaluation methodology used.

3.2 Methodological Appraisal

In general, both realism and complexity increase as you move diagonally downward and to the right in the evaluation methodology space. Note that only methods in areas A1, A2, B1 and B2 are truly replicable, since live request stream samples change over

	Workload Source		
Algorithm Implementation	artificial	captured logs	current requests
simulated systems/network	A1	A2	A3
real systems/isolated network	B1	B2	B3
real systems/real network	C1	C2	C3

Table 1: *One space of evaluation methodologies for proxies.*

time, as does the availability and access characteristics of hosts via a live network connection.

As can be seen, caching mechanisms are most commonly evaluated by simulation on captured client request logs like the sample log shown in Figure 1. These logs are generally recorded as the requests pass through a proxy, but it is also possible to collect logs by packet sniffing (as in [WWB96, CDF$^+$98]) or by appropriately modifying the browser to perform the logging (as in [TG97, CBC95, CP95]).

Simulated systems.
Simulating the caching algorithm requires detailed knowledge of the algorithms which is not always possible (especially for commercial implementations). Even then, simulation cannot accurately assess document retrieval delays [WA97].

It is also impossible to accurately simulate caching mechanisms that prefetch on the basis of the contents of the pages being served (termed *content-based prefetching*), such as those in CacheFlow [Cac99] and Wcol [CY97], since they need at least to have access to the links within web pages — something that is not available from server logs. Even if page contents were logged (as in [CBC95, MDFK97]), caches that perform prefetching may prefetch objects that are not on the user request logs and thus have unknown characteristics such as size and web server response times.

Real systems/isolated networks.
In order to combat the difficulties associated with a live network connection, measurement techniques often eliminate the variability of the Internet by using local servers and isolated networks, which may generate unrealistic expectations of performance. In addition to not taking into consideration current web server and network conditions, isolated networks do not allow for retrieval of current content (updated web pages).

Real systems and networks.
Because the state of the Internet changes continuously (i.e. web servers and intermediate network connections may be responsive at one time, and sluggish the next), tests on a live network are generally not replicable. Additionally, to perform such tests the experiment requires reliable hardware, robust software, and a robust network connection to handle the workload applied.

Artificial workloads and captured logs.
Synthetic workloads often assume that all objects are cachable. Using actual client request logs is helpful, but since on average the lifetime of a web page is short (less than two months [Wor94, GS96, Kah97, DFKM97]), any captured log loses its value quickly as more references within it are no longer valid, either by becoming inaccessible or by changing content (i.e., looking at a page a short time later may not give the same content). In addition, unless the logs are processed carefully, it is possible for the logs to reflect an inaccurate request ordering as the sample Squid logs show in Figure 1. Note that the request for the main page follows requests for three subitems of that main page. Finally, proxy cache trace logs are inaccurate when they return stale objects since they may not have the same characteristics as current objects.

Current request stream workloads.
Using a live request stream produces experiments that are not reproducible (when paired with live hardware/networks). Additionally, the test hardware and software may have difficulties handling a high real load.

Except for Wooster and Abrams [WA97], who embed multiple algorithms into the same proxy, we are aware of no work that attempts to evaluate multiple proxies at the same time. Therefore we suggest an alternative and somewhat complementary approach to web and proxy server evaluation. We propose an architecture that is online, comparing multiple proxies at once. It can use actual object requests as they are generated by a user population (column 3) and sends them to multiple proxies in parallel and any unfilled requests can be sent out over a live network connection.

```
893252015.307 14 <client-ip> TCP_HIT/200 227 GET
  http://images.go2net.com/metacrawler/images/transparent.gif - NONE/- image/gif
893252015.312 23 <client-ip> TCP_HIT/200 4170 GET
  http://images.go2net.com/metacrawler/images/head.gif - NONE/- image/gif
893252015.318 38 <client-ip> TCP_HIT/200 406 GET
  http://images.go2net.com/metacrawler/images/bg2.gif - NONE/- image/gif
893252015.636 800 <client-ip> TCP_REFRESH_MISS/200 8872 GET
  http://www.metacrawler.com/ - DIRECT/www.metacrawler.com text/html
893252015.728 355 <client-ip> TCP_HIT/200 5691 GET
  http://images.go2net.com/metacrawler/images/market2.gif - NONE/- image/gif
893252016.138 465 <client-ip> TCP_HIT/200 219 GET
  http://images.go2net.com/metacrawler/templates/tips/../../images/pixel.gif - NONE/- image/gif
893252016.430 757 <client-ip> TCP_REFRESH_HIT/200 2106 GET
  http://images.go2net.com/metacrawler/templates/tips/../../images/ultimate.jpg -
  DIRECT/images.go2net.com image/jpeg
```

Figure 1: *This excerpt of a proxy log generated by Squid 1.1 [Wes98] records the timestamp, elapsed-time, client, code/status, bytes, method, URL, client-username, peerstatus/peerhost and object-type for each request. It is also an example of how requests can be logged in an order inappropriate for replaying in later experiments, since the ordering (and timestamp) reflect request completion instead of request origination.*

The SPE architecture (detailed in the next section) is complementary to existing methodologies, but is the first mechanism to allow for the simultaneous black-box comparison of competing mechanisms on live, real-world data in a real-world environment.

4 The SPE Architecture

The SPE architecture can use actual client requests and the existing network to evaluate multiple proxies. It records timing measurements to calculate proxy latency and can compute page and byte hit rates. In summary, it forms a wrapper around competing proxies and produces logs that measure external network usage as well as performance and consistency as seen by a client. By simultaneously evaluating multiple proxies, we can utilize a single, possibly live, source of web requests and a single network connection to provide objective measurements under a real load.

4.1 Architecture Details

This paper proposes an architecture for the simultaneous evaluation of competing proxy caches. In it, web requests are funneled through an initial proxy (which we call a *Multiplier*) to replicate the request to multiple proxy caches, which each attempt to satisfy the request. To do so, each either returns a cached copy of the requested object, or forwards the request to a specialized proxy cache (termed a *Collector*) for fulfillment. The proxy Collector also attempts to return a cached copy of the requested object, but if not possible, tries to fetch it from the proper web server (or potentially ask yet another proxy cache for the document). By forcing the proxies to send their requests through the Collector, only a single request ends up being sent sent out to the Internet for fulfillment. See Figure 2 for a diagram of this architecture.

When a user of a proxy-enabled browser clicks on a link, the browser sends the request for that URL to the proxy. The proxy, in this case the Multiplier, takes the request and sends a duplicate request to each of the proxies under evaluation and directly to the Collector. Some proxies may have the object in the cache, in which case they return the object quickly. Others may experience a *cache miss*, or need to verify the contents of cache, and so have to send a request through the Collector.

Without a proxy Multiplier, client browsers would need modifications to send requests to each proxy under evaluation. Instead, one can configure off-the-shelf browsers to use the Multiplier as their proxy. In addition to request duplication, the Multiplier calculates request fulfillment times for each of the proxies being evaluated. The Multiplier also determines which response is passed back to the client — either arbitrarily, or by an algorithm such as by first response.

Each of the proxy caches being evaluated can be treated as a black-box — we do not need to know how they work, as long as they function as a valid

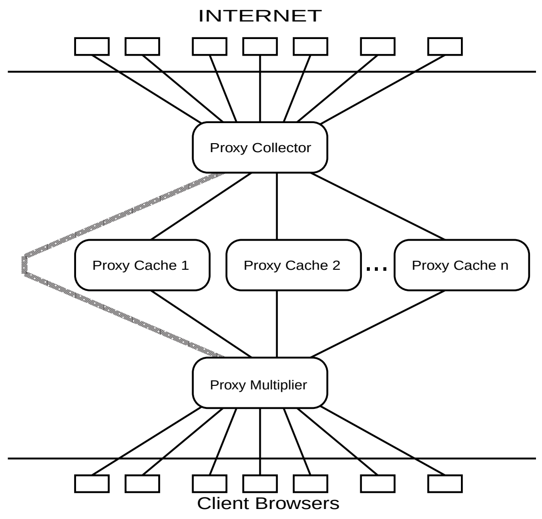

Figure 2: *The proposed architecture for online evaluation of competing proxy caches. All client requests are sent to the proxy Multiplier, which sends a copy to each proxy cache being evaluated as well as directly to the Collector. If a cached copy of the requested item is available, the proxy cache returns it, but otherwise the proxy sends the request to the proxy Collector. The proxy Collector, which is also a cache, does the same (filling requests from its cache if possible) but fetches directly when necessary.*

HTTP 1.0 [BLFF96] or 1.1 [FGM$^+$97] proxy cache. This is helpful in particular for commercial proxies, but in general eliminates the requirement for either detailed knowledge of the algorithms used or source code which is needed for simulation and specialized code additions for logging [MR97] respectively. Typically the proxy caches would run on different machines, but for simple tests that may not be necessary. Alternately, the same software with different hardware configurations can be tested in this architecture.

In order to prevent each proxy cache from issuing its own request to the destination web server, we utilize a proxy Collector which functions as a cache to eliminate extra traffic that would otherwise be generated. It cannot be a standard cache, as it must eliminate duplicate requests that are normally not cachable (such as those resulting from POSTs, generated by cgi-bin scripts, designated as private, etc.). By caching even uncachable requests, the Collector will always be able to prevent multiple requests from being issued for requests that are likely to have side effects such as adding something to an electronic shopping cart. However, the Collector must then be able to distinguish between requests for the same URL with differing arguments (i.e., differing fields from otherwise identical POST or GET requests). In addition, the Collector offers each proxy just one copy of this object from its cache, as additional requests signify a new request by the client which needs to be passed to the destination web server.

The Collector returns all documents with the same transmission characteristics as when fetched from the destination server. This includes the amount of time for initial connection establishment as well as transmission rate. By accurately simulating a "miss", the Collector 1) enables the black-box proxy to potentially use the cost of retrieval in its calculations, and 2) allows the Multiplier to record response times that more accurately reflect real-world performance. Without it, any requests for objects that are in the Collector's cache will inappropriately get a performance boost.

To summarize, traditional evaluation of caching algorithms has been via trace-based simulations. We propose an online method, useful because it:

- takes existing proxy caches and uses them as black-boxes for evaluation purposes.
- can test the proxy caches on real-world data.
- eliminates variation in measured performance due to network/server changes over time.
- eliminates variation in measured performance due to "dated" request logs.
- allows for the evaluation of content-based prefetching systems without having to locally mirror all pages referenced in a trace or to generate an artificial workload complete with internal links.
- allows for the evaluation of prefetching systems that might fetch pages never seen by the user (and thus not included in a traditional trace).
- allows for the measurement of consistency by comparing objects returned from proxies with that fetched directly.
- simultaneously compares different proxies on the same request stream.
- can evaluate cache freshness.

We realize that any method has drawbacks. We have identified the following disadvantages of the SPE architecture (with comments in *italics*):

- evaluating multiple prefetching schemes may place a higher demand for bandwidth on the network.
- multiple copies of comparable hardware may be needed. *(At least $n + 1$ systems are needed to compare n different proxy caches.)*
- the Multiplier and Collector may introduce additional latencies.
- proxy Multiplier hides the client ip (as it may be useful for a proxy to perform tasks differently depending on the client).
- caches are evaluated independently (hierarchies are not easily included, even though better performance is often claimed when part of a hierarchy).
- tests are not replicable. *(This is of course applicable to any method using a live network connection.)*
- Multiplier and Collector must manage large numbers of connections *(discussed further below).*

4.2 Implementation Issues

The SPE architecture described has characteristics that may not be entirely feasible in some circumstances. For example, to eliminate any requirements for user intervention, there should be an existing proxy that can be replaced by the Multiplier (i.e., at the same ip and port address), or the Multiplier and Collector combination could be configured to transparently handle and route HTTP requests.

In the earlier description of the SPE architecture, all HTTP requests go through all proxies when not filled by the previous proxy's cache. In particular, all black-box proxy misses go through the Collector. To ensure its use, the Collector should be placed on a firewall so that there is no chance of the proxies being tested getting around it.

To preserve client performance during the test, the Multiplier contacts the Collector directly. It also uses the object returned from the Collector as the standard against which the results of the proxies can be compared for consistency. When a proxy returns a stale object (in comparison) or an error, that can be recorded. By sending a request directly to the Collector, the Multiplier also eliminates the difficulty of determining which response to return to the client, and provides some degree of robustness in case of failure of the proxy caches being tested.

In general, any implementation of the SPE architecture should be as invisible to the participating software as possible. For example, caching connections has been shown to be a significant factor in cache performance [CDF$^+$98], so the Multiplier should support HTTP/1.0+ Keep-Alive and HTTP/1.1 persistent connections to clients for performance, but should only attempt to contact the proxies at the same level that the client contacted the Multiplier. In this way the Multiplier will attempt to generate the same ratio of HTTP 1.0 and 1.1 requests as it received. Likewise, the Collector may receive a range of request types from the proxies under evaluation depending on their capabilities. For highest fidelity, the Collector would record when it connects to destination web servers that support HTTP 1.1 and use that information to selectively provide HTTP 1.1 support for client proxy requests.

Unfortunately, the requirements for an accurate Collector are high. Until one is available, a traditional proxy cache may need to be used as a Collector, which means it would not attempt to cache the "uncachable" objects, nor return objects with accurate timing characteristics. This allows for the calculation of hit-rates but not comparative average latency values. Additionally, some proxies may not be able to handle firewalls, so they may attempt to fetch some objects directly. Therefore, instead of passing all requests to the proxies being tested, the Multiplier may need to be configured to fetch noncachable requests (cgi, forms, URLs containing "?", etc.) directly.

Other difficulties include implementing a robust enough Multiplier and Collector — for high-load measurements (such as that needed for enterprise-level proxies [MR97]), they need to be able to handle a higher load than the best black-box proxy being tested, and support k times as many connections (where k is the number of black-box proxies). In general, this difficulty is significant — under the SPE architecture, the Proxy and Collector have to manage a large number of connections. As a result, this architecture is not well suited for stress-testing proxies, but instead provides insight to their performance on non-peak workloads.

5 Representative Applications

To demonstrate the potential of the SPE architecture, we present the following scenarios. Each exhibits how some aspect of the SPE architecture can be valuable in evaluating alternative proxies.

5.1 Measuring hit-rates of black-box proxy caches

Consider first the task of comparing bandwidth reduction across multiple black-box proxy caches, including some that perform prefetching on a believable client load. Since the internal workings may not be known, simulation is not viable, and in order to evaluate the prefetching caches, a realistic workload with full content is desirable. This is a typical evaluation task for corporate purchasing but is also similar to that needed by researchers evaluating new prototypes.

Properly implemented, the SPE architecture can be used to channel all or part (with load balancing hardware) of the existing, live client request load through the multiple proxies under evaluation. Since both incoming and outgoing bytes to each proxy can be measured, the calculation of actual bandwidth savings on the current workload and network connection can be performed. Additionally, the SPE architecture is sufficient to measure the staleness of data returned by each proxy. Finally, note also that with some loss of generality, a minimal test of this sort can be made without implementing the Collector but instead test the proxies on cachable requests only (with the Multiplier fetching uncachable requests directly).

5.2 Measuring latency changes resulting from varying OS and hardware

In this task, a single proxy caching solution is used, such as Squid [Wes98]. The question being asked is that of which version of UNIX is best, or what hardware is sufficient to get the best performance for a particular organization's workload. If the various installations of Squid are identical, the application-level operations of the cache will be almost identical. Thus, the difference in performance from using an IDE vs. SCSI drives, or performance from a stock OS installation vs. one that is custom-tuned can be determined. Since SPE allows for each to receive the same request load and generate a single network load, the relative performance of each configuration can be calculated under the current loads and network capabilities of the particular organization.

5.3 Evaluating a transparent evaluation architecture

Finally, consider the problem of evaluating the overhead of a transparent evaluation architecture. In particular, given an implementation of SPE, it could be used to evaluate itself, to find out how much latency it introduces and to attempt to verify the consistency of the Collector implementation.

5.4 Use of a live vs. canned request stream

In each of the preceding cases, there is still the question of using a live or a canned request stream. Interactive prefetching systems rely on the existence of "thinking time" — the time between page requests — to provide time in which to prefetch the objects likely to be requested next. Past research [CP95, CBC95] suggests a heavy tailed distribution of thinking times with a mean of 30 seconds. Thus, captured request streams should not be replayed faster than originally recorded and artificially generated streams need to preserve the appearance of thinking time where relevant.

Ideally, one would use a live request stream to eliminate aging effects, and especially for prefetching systems. However, in some cases it may not be possible, and so in order to preserve the thinking time, it is desirable to replay a trace log faithfully. One possibility would be to take a publicly available trace log and re-run it. Unfortunately, many public trace logs hide the destination URL, rendering them useless for this purpose, and most others are fairly old, making them less reliable for live replay on the WWW. Naturally, replaying any trace log on a live network connection is dangerous unless requests generating side-effects are filtered out. Therefore, we suggest using either the current request stream, or at worst a very recent trace log from the same client base, after appropriate cleansing.

6 Conclusions and Future Research

In this paper we presented an online evaluation method, called the SPE (Simultaneous Proxy Evaluation) architecture, which simultaneously evaluates different caching strategies on the same request stream. This is in contrast to existing evaluation methodologies that use less realistic test environments and potentially out-of-date or irrelevant workloads. Importantly, it also allows for the evaluation of content-based prefetching proxies, and can test for cache consistency. We have outlined a space of evaluation methodologies and shown where SPE falls within it, and have provided some sample applications to demonstrate the potential for the proposed architecture.

The SPE architecture has been designed in particular for use in an online environment with real traffic.

However, it can still be useful as an evaluation technique with artificial workloads or trace logs as in the experiments reported here. In fact, artificial workloads are often useful for evaluating systems under workloads that cannot easily be captured or are in excess of current loads. In this way one can capture the statistical properties of specialized workloads and still retain the benefits of simultaneous evaluations on a live network.

As mentioned in the introduction, we have ongoing development effort in building a proof-of-concept prototype that implements the SPE architecture, called ROPE (Rutgers Online Proxy Evaluation). At present we have an initial Java implementation with most of the functionality of the Multiplier, but none of the Collector. As a result, we have performed very little evaluation of this incomplete implementation, but have used it in some pedagogical examples to demonstrate the feasibility of the architecture. As we continue our development, we expect to also implement a timing-sensitive Collector to be able to apply the SPE architecture in online head-to-head timing comparisons of alternative proxy implementations. We are actively looking for collaborators with the appropriate infrastructure to provide the environment in which the prototype can be tested over a reasonable time period (e.g. weeks).

The SPE architecture is particularly useful for evaluation of black-box proxies and prefetching systems. It can be used to help validate manufacturer claims and provide comparisons between systems using live networks. Evaluation systems that implement the SPE architecture will be welcome additions to the set of measurement tools available to cache designers and network engineers alike.

Acknowledgments

Thanks are due to Haym Hirsh, Brett Vickers, and anonymous reviewers for their helpful comments on earlier drafts of this paper.

References

[BLFF96] Tim Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol — HTTP/1.0. RFC 1945, `http://nic.ddn.mil/ftp/rfc/rfc1945.txt`, May 1996.

[Cac99] CacheFlow Inc. CacheFlow products web page. `http://www.cacheflow.com/products/`, 1999.

[CBC95] Carlos R. Cunha, Azer Bestavros, and Mark E. Crovella. Characteristics of WWW Client-based Traces. Technical Report TR-95-010, Computer Science Department, Boston University, July 1995.

[CDF+98] Ramón Cáceres, Fred Douglis, Anja Feldmann, Gideon Glass, and Michael Rabinovich. Web proxy caching: the devil is in the details. In *Workshop on Internet Server Performance (WISP'98)*, Madison, WI, June 1998.

[CDN+96] Anawat Chankhunthod, Peter B. Danzig, Chuck Neerdaels, Michael F. Schwartz, and Kurt J. Worrell. A Hierarchical Internet Object Cache. In *Proceedings of the USENIX Technical Conference*, San Diego, CA, January 1996.

[CP95] Lara D. Catledge and James E. Pitkow. Characterizing Browsing Strategies in the World Wide Web. *Computer Networks and ISDN Systems*, 26(6):1065–1073, 1995.

[CY97] Ken-ichi Chinen and Suguru Yamaguchi. An interactive prefetching proxy server for improvement of WWW latency. In *Proceedings of the Seventh Annual Conference of the Internet Society (INET'97)*, Kuala Lumpur, June 1997.

[Dav99] Brian D. Davison. A Survey of Proxy Cache Evaluation Techniques. In *Proceedings of the Fourth International WWW Caching Workshop (WCW99)*, San Diego, CA, March 1999.

[DFKM97] Fred Douglis, Anja Feldmann, Balachander Krishnamurthy, and Jeffrey C. Mogul. Rate of Change and other Metrics: a Live Study of the World Wide Web. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS '97)*, December 1997. An extended version is available as AT&T Labs - Research Technical Report 97.24.2.

[DMF97] Bradley M. Duska, David Marwood, and Michael J. Feely. The Measured Access Characteristics of World-Wide-Web Client Proxy Caches. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS '97)*, December 1997.

[FGM+97] R. Fielding, J. Gettys, Jeffrey C. Mogul, H. Frystyk, and Tim Berners-Lee. Hypertext Transfer Protocol — HTTP/1.1. RFC 2068, `http://nic.ddn.mil/ftp/rfc/rfc2068.txt`, January 1997.

[GS96] James Gwertzman and Margo Seltzer. World-Wide Web Cache Consistency. In *Proceedings of the USENIX Technical Conference*, San Diego, CA, January 1996.

[Kah97] Brewster Kahle. Preserving the Internet. *Scientific American*, 276(3):82–83, March 1997.

[MDFK97] Jeffrey Mogul, Fred Douglis, Anja Feldmann, and Balachander Krishnamurthy. Potential Benefits of Delta-encoding and Data Compression for HTTP. In *Proceedings of ACM SIGCOMM*, pages 181–194, September 1997. An extended and corrected version appears as Research Report 97/4a, Digital Equipment Corporation Western Research Laboratory, December, 1997.

[ML97] Jeffrey Mogul and P. Leach. Simple hit-metering and usage-limiting for HTTP. RFC 2227, `http://nic.ddn.mil/ftp/rfc/rfc2227.txt`, October 1997.

[MR97] Carlos Maltzahn and Kathy J. Richardson. Performance issues of enterprise level web proxies. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 13–23, Seattle, WA, June 1997. ACM Press.

[TG97] Linda Tauscher and Saul Greenberg. How People Revisit Web Pages: Empirical Findings and Implications for the Design of History Systems. *International Journal of Human Computer Studies*, 47(1):97–138, 1997.

[WA97] Roland P. Wooster and Marc Abrams. Proxy caching that estimates page load delays. In *Proceedings of the Sixth International World Wide Web Conference*, pages 325–334, Santa Clara, CA, April 1997.

[Wes98] Duane Wessels. Squid Internet object cache documentation. Available at `http://squid.nlanr.net/Squid/`, 1998.

[Wor94] Kurt Worrell. Invalidation in large scale network object caches. Master's thesis, University of Colorado, Boulder, CO, 1994.

[WWB96] Roland O. Wooster, Stephen Williams, and Patrick Brooks. HTTPDUMP: Network HTTP packet snooper. Working paper available at `http://www.cs.vt.edu/~chitra/work.html`, April 1996.