Thin-Client Web Access Patterns: Measurements from a Cache-Busting Proxy*

Terence Kelly
tpkelly@eecs.umich.edu
Electrical Engineering & Computer Science
University of Michigan
Ann Arbor, Michigan 48109 USA

Abstract

This paper describes a new technique for measuring Web client request patterns and analyzes a large client trace collected using the new method. In this approach a modified proxy intercepts requests and serves all responses to clients marked uncacheable, effectively disabling browser caches and allowing the proxy to record requests that would otherwise result in silent browser cache hits. WebTV Networks used a "cache-busting proxy" to collect an unusually large and detailed anonymized Web client trace in September 2000. It contains over 347 million requests for over 36 million documents by over 37,000 clients and spans 16 days. By most measures it is two orders of magnitude larger than existing Web client traces.

We compare cache-busting proxies with conventional client instrumentation and use the WebTV trace to explore browser cache performance, reference locality, and document aliasing. We present the aggregate browser cache success function (hit rate vs. cache size) of the *entire client population* and discuss design implications for memory- and bandwidth-constrained Web clients.

For the workload studied, eliminating redundant data transfers would increase browser cache hit rates by 35% to 45% over their current levels. A simple and practical technique for eliminating redundant transfers is described. Document sharing across client reference streams is so strong that the hit rate of a shared proxy cache could exceed 57% even if browser caches were infinitely large.

Keywords: World Wide Web, WebTV, client trace, browser trace, instrumentation, workload characterization, duplicate suppression, hit rates, cache simulation, success function, LRU stack distance, lognormal models.

1 Introduction

To design efficient and cost-effective systems we must understand the workloads submitted to them. World Wide Web workload consists of two fundamental components: the universe of data made available by servers, and client requests for these data. This paper deals with the latter aspect of Web workload. We discuss a new technique for measuring Web client request streams and describe how it was used to collect a large and detailed trace at WebTV Networks. The paper also presents a preliminary workload analysis, including simulation results describing the aggregate hit rate of the entire client population as a function of browser cache size. For the workload studied the potential benefits of eliminating redundant proxy-to-browser transfers are large, and we describe a simple way to obtain these benefits.

Empirical Web caching research is largely based on proxy logs. Implementors and administrators regard these logs primarily as security features; consequently the logging capabilities of most proxies are not well suited to research. Logs rarely record all of the data available to the proxy and typically omit information crucial to accurate trace-driven simulation. In particular, they fail to record cache-related HTTP metadata in reply headers and "`META http-equiv`" tags within HTML files, reply entity-bodies or their hashes, and accurate, high-resolution timestamps. Davison and Cáceres et al. have described the shortcomings of conventional proxy log formats [14, 18].

In a few cases researchers have instrumented browsers to collect Web client traces [15, 17]. In principle, such traces support arbitrarily realistic bottom-up explorations of cache hierarchies and shed light on user interactions invisible outside the client. Unfortunately, today most researchers cannot instrument popular browsers because source code is unavailable. Even if it were, it is difficult to deploy an instrumented browser among a large and representative sample of Web users. Furthermore, if such a feat were possible it would still be difficult to synchronize large numbers of client clocks, and without precise event timestamps accurate simulation of a cache hierarchy is impossible. Finally, elaborate browser instrumentation may not be an option in severely memory-constrained client devices.

Another problem with existing proxy and client traces is that most were collected in idiosyncratic environments, e.g., academic computer science departments and computer corporations. Finally, nearly all existing traces were generated by heavyweight clients: full-featured browsers running on desktop PCs or engineering workstations. They may not be representative of workloads on the memory- and bandwidth-constrained browsers that are proliferating as the Web expands onto set-top boxes and wireless handheld devices.

This paper describes a technique that combines the relative ease of proxy logging with some of the advantages of client instrumentation. In this method a "cache-busting proxy" intercepts requests from unmodified clients and labels all replies uncacheable, thereby disabling browser caches and allowing the proxy to log requests that would otherwise be served silently from

*Proceedings of the Sixth International Workshop on Web Caching and Content Distribution, 20–22 June 2001, Boston University, Boston, Massachusetts, USA. `http://cs-pub.bu.edu/pub/wcw01/`
The most recent version of this paper is available at
`http://ai.eecs.umich.edu/~tpkelly/papers/`
`$Id: wtvwl.tex,v 1.50 2001/06/12 02:35:40 tpkelly Exp $`

browser caches. An informal survey of Web researchers reveals that this technique has been proposed before; it was discussed by a group at Boston University in late 1999 [13] and is described in a recent book by Krishnamurthy & Rexford [24]. However to the best of our knowledge it has never before been used.

In September 2000 WebTV Networks collected a large anonymized trace of client accesses using a cache-busting proxy. The proxy itself ran in non-caching mode; the trace therefore reflects activity in a cacheless system. The proxy furthermore recorded a checksum of every entity-body (data payload) received from origin servers, as well as a checksum of the (possibly different) entity-body served to the client after transcoding by the proxy. All events in this trace are timestamped at microsecond resolution by well-synchronized proxy clocks. The proxy recorded all cache-related HTTP metadata in client requests, server reply headers, and "`META http-equiv`" tags in HTML files. WebTV's trace spans 16 days and records over 347 million requests to over 36 million documents by over 37,000 clients; it is two orders of magnitude larger than any client trace described in the Web caching literature.

This paper presents a preliminary analysis of the WebTV trace and describes how it was recorded. Careful capacity planning based on workload measurement is essential in large-scale deployments of spartan clients, where neither storage nor bandwidth are cheap or abundant. Therefore the relationship between browser cache size and performance is our primary concern. An efficient single-pass simulation algorithm permits us to compute arbitrarily-weighted hit rates at *every* cache size for *each* client in the WebTV trace [23]. It is straightforward to aggregate individual client success functions to obtain hit rate as a function of cache size for the entire client population, and this is the centerpiece of our analysis. Reference locality and document aliasing have performance implications, and we shall explore these issues as well.

Our main finding is that browser cache hit rates would increase substantially if redundant payload transfers resulting from unnecessary cache misses were eliminated, and we describe a simple and practical way to eliminate redundant transfers.

2 Related Work

Researchers have investigated in detail the workload placed on *components* of the World Wide Web, e.g., servers, proxies, and networks [4–9, 19–21, 38, 39]. However, little is known about the workload placed on the Web *as a system*, i.e., the universe of available documents and patterns of client requests. Recently Padmanabhan & Qiu have investigated content creation and modification dynamics at a large, busy Web site [34]; this is the only systematic study of available content (not to be confused with *requested* content, i.e., server access patterns) of which we are aware.

The situation is only slightly better at the client end, where true client traces—request streams not filtered by browser caches—are extremely rare. The Web Characterization Repository [1] contains several proxy and server workloads but only a single client trace, collected at Boston University's Computer Science Department in 1995. Catledge & Pitkow recorded a client trace at Georgia Tech's CS department in 1994. Both the BU and Georgia Tech traces are remarkably rich, recording a wide variety of user-interface events unavailable outside the browser. Together these two traces have supported a number of interesting studies [12, 15–17]. Unfortunately, as Netscape and Microsoft Internet Explorer displaced the open-source Mosaic browser in the late 1990s it became difficult for researchers to instrument Web browsers, and no true client traces have been collected since 1995.[1]

		cache size		# in	H.R.
Type	Description	RAM	disk	trace	(%)
FCS	"Classic"	420 KB		8,790	37.64
BPS	No-frills	1240 KB		11,253	41.94
LC2.5	"Plus"	3200 KB		7,370	44.41
LC2	Diskful Plus	1 MB	20 MB	8,535	44.64
ST1	Satellite	3 MB	20 MB	1,221	44.81

Table 2: WebTV client devices.

Table 1 summarizes the aforementioned Web client traces, more recent proxy and server traces used in References [6, 7, 28, 39], an AFS client trace [32] and the WebTV trace that is the subject of this paper. The most striking feature of Table 1 is the large size difference between the early client traces and the more recent proxy and server traces; by nearly every measure the latter are orders of magnitude larger. The difficulty of deploying an instrumented browser on large numbers of clients is largely responsible for the difference.

To summarize, the few existing Web client traces are several years old, reflect the requests of computer science students, and are small in comparison with server and proxy traces. By contrast server and proxy traces are often large and sometimes describe more representative user populations but typically omit much information in the original client request streams, e.g., references served from browser caches. Section 3 describes a collection methodology that combines some of the advantages of client and proxy traces and explains how this technique was used to measure Web client workload on a very large scale.

3 Trace Collection

With over a million active subscribers WebTV Networks is among the largest ISPs, and its customer base is arguably more representative of the general public than the traditional subjects of Web traces (computer science students and computer industry employees). Furthermore the WebTV system is extraordinarily well integrated, providing essentially everything but the origin server: client hardware, browser software, proxies, and Internet connectivity. Most importantly, WebTV staff constantly monitor and tune the system to improve its performance, frequently adding new instrumentation as new questions arise. For these reasons WebTV is an ideal environment for Web-related research.

WebTV clients represent an interesting intermediate point in design space, midway between the resource-rich PC-based browsers of the early Web and the ultra-thin clients of tomorrow. WebTV employs a relatively inexpensive (often diskless) set-top box to enable Web surfing on a conventional television. Five types of client devices were in use during September 2000; see Table 2. Clients connect to the WebTV service via modem; according to WebTV's measurements, bandwidth to clients varies but is typically roughly 33.6 Kbps.

[1] A 1999 sequel to the original B.U. study used a trace that did not reflect browser cache hits [10].

Trace	Type	Begin	End	Clients	Objects	Requests	Requests per Client per Day
CITI AFS	client	20 Oct 93	20 Dec 93	37	N/A	12,192,933	5,402
Georgia Tech.	client	3 Aug 94	24 Aug 94	107	9,452	43,060	19
Boston U.	client	21 Nov 94	17 Jan 95	600	46,830	575,775	17
Cable Modem	proxy	3 Jan 97	31 May 97	$\approx$ thousands	16,110,126	117,652,652	
World Cup	server	1 May 98	23 Jul 98	2,770,108	20,728	1,352,804,107	6
Compaq WRL	proxy	1 Jan 99	31 Mar 99	$\approx$ 25,000	N/A	125,259,641	54
U. Washington	proxy	7 May 99	14 May 99	22,984	$\approx$ 18,400,000	$\approx$ 82,800,000	515
Microsoft	proxy	7 May 99	14 May 99	60,233	$\approx$ 15,300,000	$\approx$ 107,700,000	286
WebTV	client	7 Sep 00	22 Sep 00	37,169	36,573,327	347,483,200	425

Table 1: Traces used in Web and file system research. Dynamic IP addresses prevented a precise count of clients in the Cable Modem trace [3]. The number of distinct documents is not known for the Compaq data [27].

Our original goal was to collect a client trace using instrumented browsers. WebTV frequently downloads software updates to its client devices, so at first this seemed a straightforward approach. However a combination of logistical difficulties and schedule constraints forced us to seek an alternative approach. Compared with client software modifications, proxy patches are much easier to implement and deploy and are far more frequent in practice. We therefore decided to record unfiltered client requests by disabling the browser cache with a modified proxy.

A sophisticated centralized service infrastructure compensates for WebTV clients' limitations by transcoding images, re-writing HTML, and maintaining persistent state (e.g., cookies). Sixteen modified proxies collected WebTV's client trace, with the following non-standard features enabled during data collection:

- events were timestamped at microsecond resolution;
- checksums were logged of all entity bodies received from origin servers and all entity bodies served to clients after transcoding;
- all metadata relevant to caching in client requests, server reply headers, and embedded HTML tags were logged;
- all documents were served to clients with an "`Expires: 0`" HTTP header;[2]
- the proxy itself was run in non-caching mode.

During normal operations WebTV proxies process every byte of every data payload, so the performance penalty of computing checksums was relatively minor; similarly, the proxies normally parse META tags in HTML. The number of proxy hosts that collected data was twice as large as would normally be used for our sample client population, so the proxies were not overloaded as they recorded the trace. All trace fields related to user identity and requested content are anonymized to protect user privacy.

META tags are necessary for correct browser cache simulations and are furthermore interesting because they illustrate discrepancies between HTML-embedded metadata and HTTP headers. For instance, META expiration dates often disagree with HTTP expirations. Of over 149 million requests issued during 16–22 September, 95,558 have expiration dates in both headers and META tags. Among these requests the META expiration is earlier than the response header expiration 21.8% of the time; ignoring the HTML-embedded expiration could cause consistency violations in some of these cases. In 141,159 cases an expiration is specified only in a META tag; unnecessary revalidations would result from ignoring the embedded metadata in some of these cases.

[2]Caching pre-expired objects is legal in HTTP/1.1 [22], but WebTV clients do not do so.

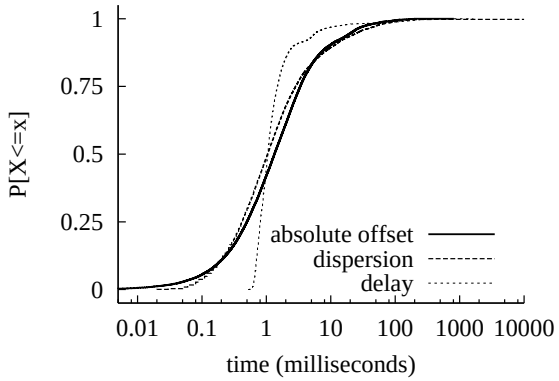

Figure 1: Distribution of NTP parameters during data collection.

WebTV proxy host clocks are carefully synchronized using Network Time Protocol [26]. A script queried the proxies with `ntpq` at ten-minute intervals throughout the data-collection period to check their clock synchronization; Figure 1 shows the distribution of observed NTP parameters. The absolute offset of the proxies with respect to an accurate reference is nearly always under 10 milliseconds.

A sample of client devices was "attached" to our modified proxy bank throughout the data collection period, i.e., all Web sessions initiated by these devices were handled by the special proxies. WebTV clients communicate directly with origin servers for secure transactions, which are therefore not reflected in our trace. Furthermore the trace contains only HTTP requests; we did not record the small volume of FTP and Gopher traffic handled by the proxies.

We began serving documents pre-expired on 6 September 2000. The number of requests per hour reaching our proxies promptly doubled, as illustrated in Figure 2. By comparing request volumes from each client device type before and after 6 September we obtain crude estimates of browser cache hit rates; estimates based on requests of 4–5 vs. 11–12 September are shown in Table 2. This technique must be used with care, because our sample browser caches did not cease to operate on 6 September, they merely ceased to cache incoming documents. It is reasonable to suppose that many browser caches remained "warm" even after our experiment began, serving some fraction of requests from cache. We gain insight into browser

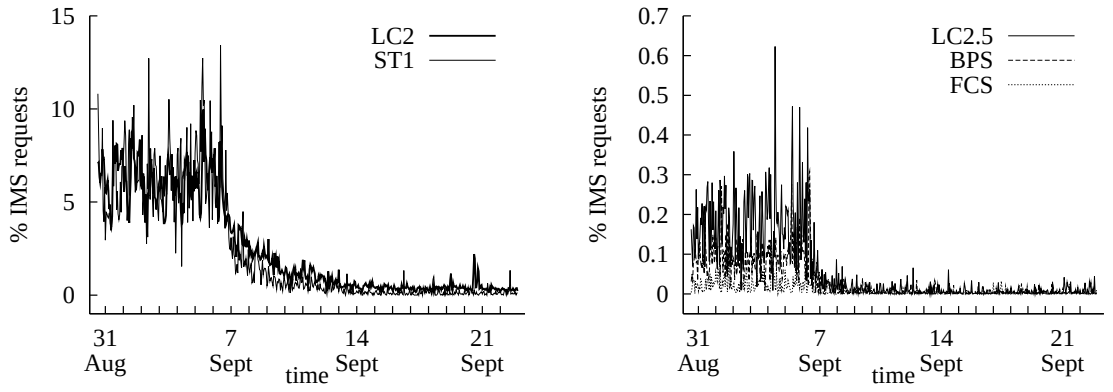

Figure 3: Percentage of IMS requests from diskful (left) and diskless (right) clients, 1-hour bins. Vertical scales differ.

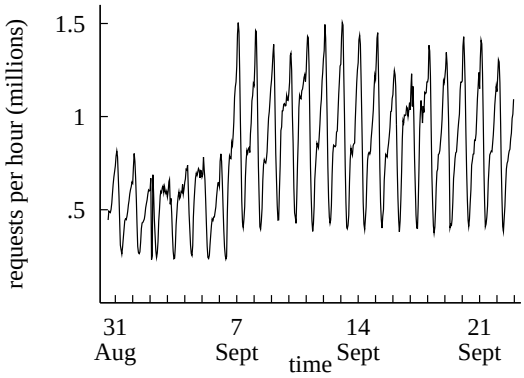

Figure 2: Hourly request volume

Clients		37,169
Payloads (entity-bodies)		36,573,327
Total requests		347,483,200
Successful requests		325,591,889
Sum of payload sizes		640,177,065,087
Bytes transferred*		4,376,249,806,158
	mean	median
Payload size	17,504	5,493
Transfer size*	13,441	2,916
Requests per client*	8,760	3,326

Table 3: WebTV trace summary statistics. All sizes in bytes. Asterisk (*) indicates figures based on successful requests only.

cache "cool down" from the percentage of revalidation requests ("`If-Modified-Since`") reaching our proxies from each device type over time, as shown in Figure 3.

Rich-client browsers typically use separate regions of memory to hold the currently-viewed document and to cache previously-viewed objects; the contents of the cache therefore change only when new items are cached. In the memory-constrained WebTV client, however, the same region of memory is used as both a "staging area" for the current item and as cache for previously-requested items. Large incoming documents therefore cause cache evictions *even if they are not cached themselves*. This accounts for the rapid decrease in IMS requests in Figure 3. Collectively, the browser caches in our sample are never flushed completely; even after two weeks IMS requests reach our proxies, possibly from clients with low activity levels. However the fraction of IMS requests quickly falls to negligible levels, particularly for diskless clients. Our estimates of browser cache hit rate are based on proxy request volumes several days after cache busting began, by which time most browser caches have cooled substantially. However it is possible that our estimated hit rates for diskful clients are slightly low.

In retrospect, we realize that we might have obtained better results from a more thorough cache-busting proxy that attempted to forcibly flush browser caches, e.g., by serving an HTML file with a large number of newline characters appended to it. Alternately, we might have served replies with expiration dates equal to the current time plus one second; we would then see IMS requests in most cases that would otherwise be browser cache hits.

Table 3 summarizes the WebTV trace. As noted above, it is roughly as large in most respects as recent proxy traces and substantially larger than mid-90s Web client traces. Furthermore it reflects a client sample comparable to the entire end-user population served by NLANR's cache hierarchy, which is thought to be under 100,000.[3] The compressed trace occupies roughly 101.6 GB on disk.

4 Preliminary Analysis

A thorough workload characterization of the WebTV trace is the subject of ongoing research; this section presents only a brief preliminary analysis. For simplicity and also for computational reasons we focus on the entity-bodies returned by origin servers rather than the URLs and request headers submitted by clients. For brevity we shall often use the terms "document" or "payload" instead of "entity-body." Throughout this section we consider only successful requests (response code 200); the trace records over 325 million such events (93.7% of the total). We equate a document with its payload checksum and define its size to be the largest document-size-related reply header associated with it; size therefore includes protocol overhead. Figure 4 shows the distributions of document and transfer sizes.

Figure 5 shows the percentage of replies containing never-before-seen payloads in non-overlapping windows of 10,000,000

[3] NLANR caches' `x-forwarded-for` logs recorded 98,144 unique "leaf" IP addresses during the first 26 days of October 2000 [37]. This may over-estimate the end-user population due to the use of dynamic IP addresses.

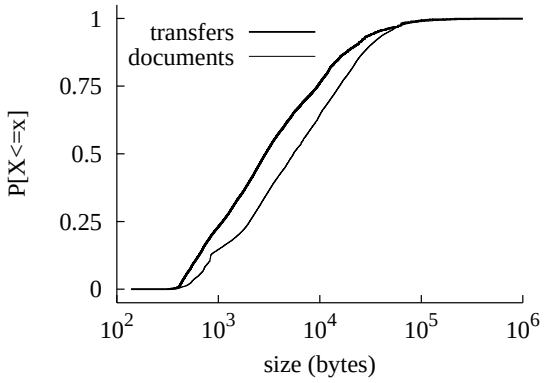

Figure 4: Distribution of document and transfer sizes.

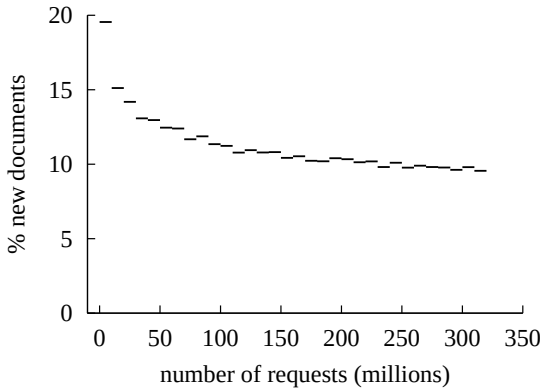

Figure 5: Percentage of replies containing new documents.

requests for the first 320 million references in the WebTV trace. This is identical to the minimal miss rate of the overall WebTV system, assuming caches so large that capacity misses never occur. The figure shows that in the absence of redundant payload transfers the steady-state hit rate of an infinitely large WebTV proxy serving cacheless clients exceeds 90%; even a cold proxy cache would enjoy an 80% hit rate. In practice, imperfect cache consistency mechanisms and namespace complexities (aliasing) cause unnecessary cache misses and redundant payload transfers. Section 5 describes a simple and practical way to eliminate these problems entirely and raise hit rates to the full potential suggested by Figure 5.

Client activity levels span a wide range. Figure 6 shows the distribution of the number of requests and total bytes transferred to clients in the WebTV trace. Graphical evidence suggests that these distributions are roughly lognormal, i.e., the logarithm of requests-per-client and of bytes-per-client appears to be roughly Gaussian, as illustrated in Figure 7. The figure shows histograms of log-transformed data superimposed over a normal curve defined by sample mean and sample variance.

Similarly, graphical evidence suggests that the distribution of LRU stack distances at the client may be lognormal (Figure 7, right; the data shown are from a sample of requests from a subset of BPS clients, described below). This distribution is closely related to the success function of an LRU cache [23, 25] and is often used to measure temporal locality in reference streams [2, 9]. Almeida et al. report that references reaching *servers* appear to have lognormal stack distance distributions, and that lognormal

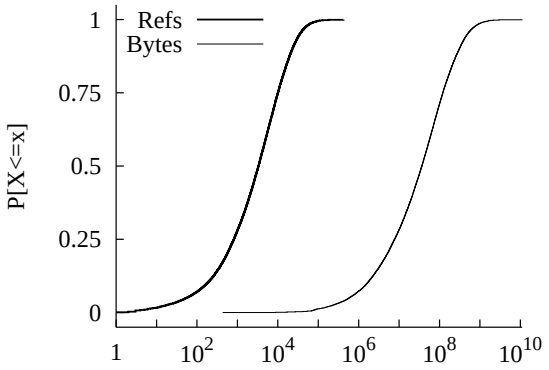

Figure 6: Distribution of references/client and bytes/client.

stack distance models predict cache success functions well [2]. We have not yet tested the usefulness of lognormal models in predicting hit rates for the WebTV data.

We might expect bandwidth-constrained thin clients to "surf" at different rates than conventional rich-client browsers in academic or corporate environments. Figure 8 shows the distribution of inter-reference intervals for the last seven days of the WebTV trace and for the Boston University client trace. WebTV requests corresponding to user-initiated actions are marked as "primary" in the trace, and the distribution of intervals between primary references is plotted separately for WebTV. The Boston distribution is bi-modal due to browser cache hits (compound objects, e.g., HTML pages with embedded images, are *not* responsible; such objects are present in both traces). The WebTV data reflect a cacheless low-bandwidth environment, and therefore it is somewhat surprising that WebTV browsers appear to be operating roughly as fast as Xmosaic: 89.5% of BU requests are served in 10 seconds or less; for WebTV the figure is 93%.

Figure 9 shows the distribution of maximal browser hit rates under ideal conditions for the observed request sequences, and the distribution of browser cache sizes required to achieve maximal hit rates. "Ideal conditions" means that the first request that yields a given payload is a miss, but all subsequent requests that would return the same payload are hits. In other words, no redundant transfers occur, and only compulsory misses occur. This is similar to the "perfect coherency" cache considered by Mogul [28–30], but it assumes no misses due to aliasing. In Mogul's terminology, we simulate a "perfect duplicate suppression" cache large enough to store all requested documents. We see from the left-hand subfigure that the median of maximal individual browser hit rates is roughly 65%.

A browser cache attains maximal hit rate if it can store all requested documents; the sum of distinct payload sizes is therefore termed the "infinite cache size" of a request sequence. However if we assume LRU replacement we can compute the maximal *priority depth* across references in a workload [23]; this is the smallest LRU cache size that experiences no capacity misses. The distribution of infinite cache sizes and maximal LRU priority depths is shown on the right of Figure 9. For the workloads studied an 11.6 MB LRU cache is effectively infinite for half of clients.

Muntz and Honeyman report that hit rates of shared intermediate caches in network file systems will not exceed 20% if reference streams are filtered by even small client caches [31]. The WebTV data, however, point to a dramatically different conclusion: Assuming *infinite* browser caches and perfect duplicate

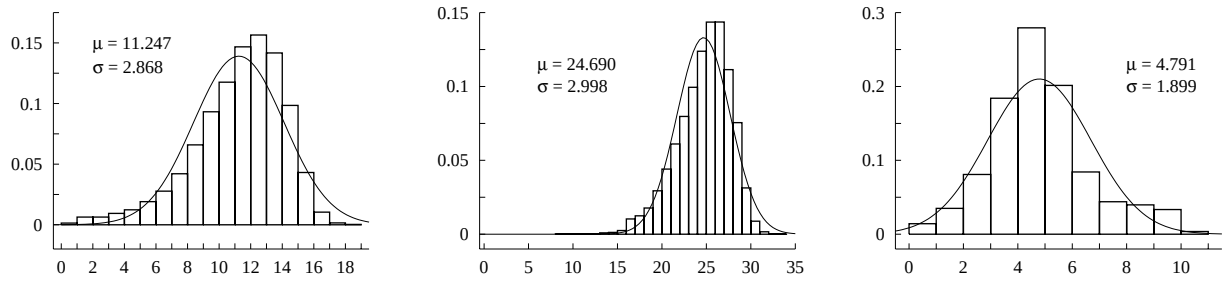

Figure 7: Histograms of $\log_2$-transformed data: *Left:* references/client. *Center:* bytes/client. *Right:* stack distances in BPS sample.

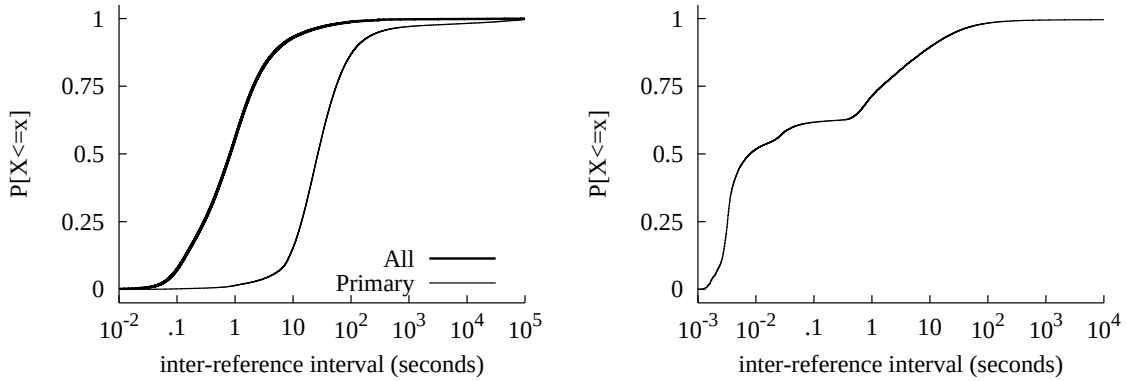

Figure 8: Distribution of inter-reference intervals in WebTV (left) and Boston University (right) client traces.

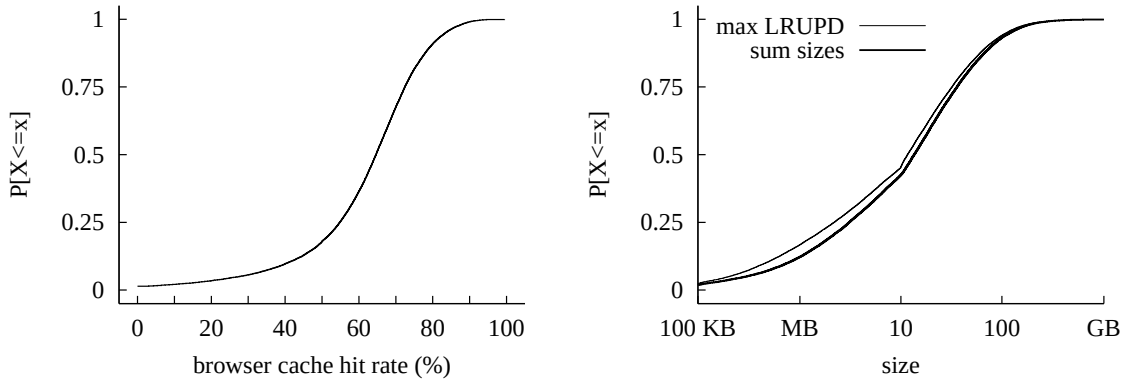

Figure 9: Distribution of maximal browser hit rates (left) and effectively infinite browser cache sizes (right).

suppression, 73.4% of requests would be served from browser caches. Of the remaining requests, 57.7% could be served from a sufficiently large shared proxy cache.

We obtain a complete picture of the relationship between browser cache size and hit rate by computing each client's success function (hit rate as a function of cache size) separately, assuming LRU replacement. We now permit capacity misses, but as before no redundant transfers occur. Efficient single-pass simultaneous simulation algorithms for this computation have long been available for the special case where document sizes and miss penalties are uniform [11, 33, 36] and have recently been generalized to non-uniform sizes and miss costs [23]. Using the Reeves-Kelly algorithm we first compute browser cache hit rates for each client at every cache size. We then aggregate the results into a single success function for the entire client population.[4]

To avoid the confounding effects of cache cool-down (Figure 3) and cold-start, we also perform the same exercise for a sample of 1,959 modern diskless (BPS) clients with moderately heavy request volumes (between median and 75th percentiles) and moderate locality (maximal browser cache hit rates between the 25th and 75th percentiles). We use each client's first 2,000 references

[4]Kelly & Reeves note that fast simultaneous simulation yields correct results only for cache sizes no smaller than the largest document in a trace [23]. This is true for rich-client browsers such as Netscape and IE, in which replies larger than the cache do not alter its contents. The memory-constrained WebTV browser, however, uses the same region of memory as both a cache and a staging area for the document currently being viewed. A reply larger than the cache will therefore flush the browser cache's contents. Stack methods can simulate WebTV-like browser caches at *all* cache sizes.

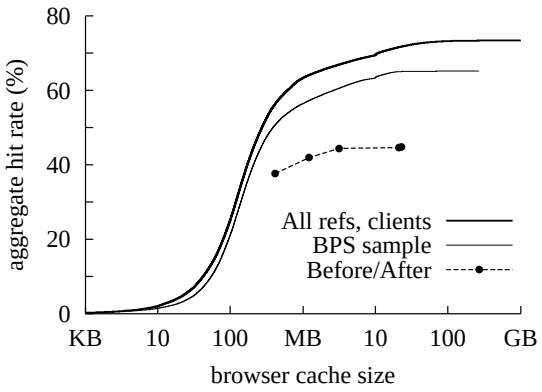

Figure 10: Aggregate WebTV client success function.

	Cache	Measured	BPS sample		All clients	
Type	Size	H.R. (%)	H.R.	R.I.	H.R.	R.I.
FCS	420 KB	37.6	50.8	35.1	56.4	50.0
BPS	1240 KB	41.9	57.3	36.8	64.1	53.0
LC2.5	3200 KB	44.4	60.6	36.5	66.9	50.7
LC2	21 MB	44.6	65.1	46.0	71.7	60.8
ST1	23 MB	44.8	65.1	45.3	71.8	60.3

Table 4: Percent relative improvement (R.I.) over current hit rates.

to warm the browser cache and tabulate hit rates based only on its next 1,000 requests. Results for both the BPS sample and the entire client population are shown in Figure 10; the estimates of actual WebTV browser hit rates from Table 2 are included for comparison. Our results are comparable to the success function presented in Figure 5 of Bestavros et al., which assumes LFU replacement [12].

Aggregate browser cache success functions are needed in order to make informed tradeoffs between browser functionality and cache hit rates in thin-client systems such as WebTV. New versions of browser software support new features and therefore require more resources, e.g., RAM, but capacity expansion is not possible in the installed base of client devices. One option for browser designers is to steal application memory from the cache. However it is impossible to know the performance implications of such a decision without browser cache success functions obtained through measurement (Table 2) or simulation (Figure 10).

Table 4 summarizes the relative improvement in aggregate browser cache hit rates that would result from eliminating redundant payload transfers. For each WebTV device type, the table compares simulated hit rates with estimates based on proxy request volumes before and after cache-busting began. The large gap between the simulated and measured success functions in the WebTV data is due to unnecessary misses caused by inefficient consistency mechanisms and aliasing. Section 5 describes a simple and practical way to completely eliminate redundant payload transfers, raising actual cache hit rates to the full potential illustrated by our simulations.

A detailed investigation of aliasing in the WebTV trace is the subject of ongoing work, and presents formidable computational challenges: Mogul reports that 7,400 MB of RAM is required to analyze duplication in a trace containing 80 million events [29]; the WebTV trace is over four times larger. As a first step, we computed the number of payloads in the trace that are aliased, i.e., that are referenced through more than one URL. Roughly 4.98% of payloads are aliased.

	Before		After	
MIME type	number	%	number	%
image/gif	43,941,647	48.11	96,904,786	64.49
image/jpeg	17,889,601	19.59	26,037,449	17.33
text/html	14,150,501	15.49	14,397,016	9.58
(missing)	13,290,676	14.55	9,469,181	6.30
application/ x-javascript	934,997	1.02	1,652,086	1.10
OTHER	1,131,681	1.24	1,795,913	1.20
TOTAL	91,339,103	100.00	150,256,431	100.00

Table 5: Most common MIME types before (31 Aug–6 Sept) and after (14–20 Sept) cache busting.

Table 5 shows the relative frequency of the twenty most common MIME types observed during one-week periods before and after cache-busting was activated. Several popular types, e.g., PDF, are absent entirely because WebTV browsers do not support them. The large increase in the popularity of GIFs may occur because GIFs are normally extraordinarily cacheable. It is also conceivable that the browser requests multiframe GIFs each time they are displayed; at the time of writing it is unclear whether WebTV browsers do so.

5 Discussion

The main contributions of this paper are a new method for recording Web browser requests, including browser cache hits, and a preliminary analysis of workload data collected with it.

The advantages of the cache-busting proxy technique are that it requires no client modifications and relatively minor proxy modifications. In most situations is it probably easier to use than conventional client instrumentation, especially for collecting large and representative data sets. However the method is not without limitations and disadvantages. One limitation is that it fails to record user-interface events and therefore cannot support the same range of investigations as an instrumented client, which remains the "gold standard" for trace collection. Furthermore the new method provides event timestamps recorded at the proxy, and these do not necessarily reflect the latency experienced by clients. If documents are served pre-expired as in the WebTV procedure, browsers may continue to serve some requests from cache even after cache-busting begins (Figure 3), so the technique does not provide a perfect record of client requests. Finally, it is unclear whether cache busting alters user behavior by increasing latency. Nonetheless a cache-busting proxy trace sheds far more light on client access patterns than ordinary proxy logs.

The most interesting result of our workload analysis is the large gap between actual and potential browser cache hit rates for the client population as a whole (Figure 10). One way to think about the simulated success functions is that they describe performance in a "trivial namespace" Web, in which a simple one-to-one correspondence exists between URLs and data payloads. To an extent such a namespace is achievable in practice, e.g., by embedding payload checksums in URLs. Content delivery networks do this already: Akamai URLs contain partial MD5 checksums of data payloads. If requests contain payload checksums, replies never

expire and stale content is never served from cache; a simple namespace improves the both the performance and the correctness of CDNs. Unchanging and mnemonic document names are necessary in systems like the Web, however, and this requirement precludes an entirely flat namespace.

Several practical techniques have been proposed to avoid redundant payload transfers, but published "duplicate suppression" schemes for the Web do not completely eliminate the problem and require that modified servers supply hints to clients. Mogul reports that one such proposal would increase an infinite proxy's hit rate and byte hit rate by 5.4% and 6.2% respectively [29, 30].

While the benefits of imperfect duplicate suppression for infinite proxy caches may be modest, WebTV's data show that the benefits of *perfect* duplicate suppression for *finite browser* caches are substantial. Our most conservative simulation data suggest that browser cache hit rates would increase by 35% to 45% over their estimated current levels (Table 4). Fortunately a simple and practical procedure can completely eliminate redundant proxy-to-client transfers in systems such as WebTV:

- The browser caches every data payload it receives, without exception.
- The browser issues ordinary requests to the proxy.
- Before the proxy returns a payload it first sends the payload's checksum.
- The browser compares the checksum to those of items in its cache. A match ends the transaction; otherwise the proxy transmits the full payload.

Several variants of this overall approach are possible: The proxy could transmit a full reply preceded by a payload checksum and halt the transmission at the client's request. Alternately the proxy might wait for an explicit "proceed/abort" from the client. The former entails no user-latency penalty and does not throttle the proxy or prolong transactions. However it may have little impact in high-bandwidth, high-round-trip-time environments if documents are small (the full payload reaches the client before its "halt" message reaches the proxy). The latter variant introduces an additional RTT into the transaction but completely eliminates redundant transfers; it may be attractive in low-bandwidth, low-RTT environments.

For purposes of duplicate suppression the proposed approach entirely ignores the namespace (URLs, request headers, etc.) and consistency mechanisms. Unnecessary misses due to aliasing and inappropriate metadata therefore disappear completely; only compulsory and capacity misses occur. The proposed approach could use entity tags as well as payload digests to avoid still more unnecessary transfers, and furthermore could be used as a hop-by-hop mechanism between any two levels in a cache hierarchy. The semantics of existing protocols, e.g., HTTP, are largely unchanged if cached payloads and digests thereof are used to avoid redundant transfers. The above scheme is similar in spirit to Spring & Wetherall's method of eliminating redundant network traffic [35]. Mogul independently conceived a technique essentially identical to the one presented here but has not published it [27].

Re-writing the rules of browser/proxy interaction is difficult in the most general case because different vendors' products must interoperate during migration and backward compatibility with legacy software must be maintained. However in more tightly integrated environments where a single organization controls both browser and upstream service infrastructure, e.g., WebTV and AOLTV, such changes are feasible. These also happen to be bandwidth-constrained environments, where the prospect of transferring compact message digests rather than far larger entity-bodies is particularly attractive.

Acknowledgments

Literally dozens of WebTV personnel helped to collect the anonymized trace described in this paper. My manager, Stuart Ozer, lent resources and his expertise to the project. Arnold de Leon provided a large computer and with Jeff Allen worked out a thousand operational details. Jay Logue modified WebTV's proxy software to log additional data and serve documents pre-expired. Todd Stansell and Jonathan Tourtellot assembled a 1.2-TB storage array and Brad Whisler managed day-to-day data logging. Paul Roy allowed us to run our measurements on a large fraction of WebTV's production system, and Andrea Chien conducted preliminary tests. Jake Brutlag helped me to understand WebTV's complex service architecture and to document WebTV's proxy log format. WebTV client engineers explained the various client devices and the rationale behind their design; Scott Sanders, Monique Barbanson, David Conroy and Wiltse Carpenter were particularly helpful. Finally, Eric Horvitz of Microsoft Research introduced me to the extraordinary research potential of WebTV.

Professor Peter Honeyman guided and funded my research. Jim Rees, Chuck Lever, and Brad Quinn of CITI designed and built a large storage array for me. The U-M Center for Parallel Computing provided computational resources that made my analyses possible; Tom Hacker was particularly helpful and flexible. Jeff Mogul provided valuable guidance on the logging of HTTP metadata. Jim Gray, Geoff Voelker, Martin Arlitt and the anonymous reviewers supplied valuable feedback on early drafts. Duane Wessels provided data on the number of clients served by the NLANR cache hierarchy. Lara Catledge and James Pitkow provided the Georgia Tech client trace, and Boston University supplied the BU trace.

References

[1] Web Characterization Repository. `http://researchsmp2.cc.vt.edu/cgi-bin/reposit/index.pl`.

[2] V. Almeida, A. Bestavros, M. Crovella, and A. de Oliveira. Characterizing reference locality in the WWW. In *4th Int'l Conf. on Parallel and Distributed Information Systems*, Dec. 1996.

[3] M. Arlitt. Personal communication.

[4] M. Arlitt, L. Cherkasova, J. Dilley, R. Friedrich, and T. Jin. Evaluating content management techniques for Web proxy caches. Technical Report HPL-98-173, HP Labs, Mar. 1999.

[5] M. Arlitt, L. Cherkasova, J. Dilley, R. Friedrich, and T. Jin. Evaluating content management techniques for Web proxy caches. In *2nd Workshop on Internet Server Performance*, May 1999.

[6] M. Arlitt, R. Friedrich, and T. Jin. Workload characterization of a Web proxy in a cable modem environment. Technical Report HPL-1999-48, HP Labs, 1999.

[7] M. Arlitt and T. Jin. Workload characterization of the 1998 World Cup Web site. Technical Report HPL-1999-35R1, HP Labs, Sept. 1999.

[8] M. Arlitt and C. Williamson. Web server workload characterization: The search for invariants. In *SIGMETRICS*, May 1996.

[9] M. F. Arlitt and C. L. Williamson. Internet Web servers: Workload characterization and performance implications. *IEEE/ACM Trans. Networking*, 5(5):631–644, Oct. 1997.

[10] P. Barford, A. Bestavros, A. Bradley, and M. Crovella. Changes in Web client access patterns: Characteristics and caching implications. *World Wide Web Journal*, 1999.

[11] B. T. Bennett and V. J. Kruskal. LRU stack processing. *IBM J. Res. Dev.*, 19(4):353–357, July 1975.

[12] A. Bestavros, R. Carter, M. Crovella, C. Cunha, A. Heddaya, and S. Mirdad. Application-level document caching in the Internet. In *2nd Int'l Workshop on Services in Distributed and Networked Environments*, June 1995.

[13] A. Bradley. Personal communication.

[14] R. Cáceres, B. Krishnamurthy, and J. Rexford. HTTP 1.0 logs considered harmful. W3C Web Char'n Group Workshop, Nov. 1998.

[15] L. D. Catledge and J. E. Pitkow. Characterizing browsing strategies in the World-Wide Web. *Computer Networks and ISDN Systems*, 27(6):1065–1073, Apr. 1995.

[16] M. E. Crovella and A. Bestavros. Self-similarity in World Wide Web traffic: Evidence and possible causes. *IEEE/ACM Trans. Networking*, 5(6):835–846, Dec. 1997.

[17] C. R. Cunha, A. Bestavros, and M. E. Crovella. Characteristics of WWW client-based traces. Technical Report BU-CS-95-010, Boston U. CS Dept., July 1995.

[18] B. D. Davison. Web traffic logs: An imperfect resource for evaluation. In *9th Conf. Internet Society*, June 1999.

[19] F. Douglis, A. Feldmann, B. Krishnamurthy, and J. Mogul. Rate of change and other metrics: a live study of the World Wide Web. In *USENIX Symposium on Internet Technologies and Systems*, pages 147–158, Dec. 1997.

[20] B. M. Duska, D. Marwood, and M. J. Feeley. The measured access characteristics of World-Wide-Web client proxy caches. In *USENIX Symposium on Internet Technologies and Systems*, pages 23–35, Dec. 1997.

[21] A. Feldmann. Continuous online extraction of HTTP traces from packet traces. In *Proc. W3C Web Char'n Group Workshop*, 1999.

[22] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. RFC 2616: Hypertext transfer protocol—HTTP/1.1, June 1999.

[23] T. Kelly and D. Reeves. Optimal Web cache sizing: Scalable methods for exact solutions. *Computer Communications*, 24:163–173, Feb. 2001.

[24] B. Krishnamurthy and J. Rexford. *Web Protocols and Practice*. Addison-Wesley, May 2001.

[25] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger. Evaluation techniques for storage hierarchies. *IBM Systems Journal*, 9(2):78–117, 1970.

[26] D. L. Mills. RFC 1305: Network time protocol, Mar. 1992.

[27] J. Mogul. Personal communication.

[28] J. C. Mogul. Errors in timestamp-based HTTP header values. Technical Report 99/3, Compaq WRL, Dec. 1999.

[29] J. C. Mogul. A trace-based analysis of duplicate suppression in HTTP. Technical Report 99/2, Compaq WRL, Nov. 1999.

[30] J. C. Mogul. Squeezing more bits out of HTTP caches. *IEEE Network*, pages 6–14, May/June 2000.

[31] D. Muntz and P. Honeyman. Multi-level caching in distributed file systems. Technical Report 91-3, U. Michigan Ctr. for IT Integration (CITI), Aug. 1991.

[32] D. A. Muntz, P. Honeyman, and C. J. Antonelli. Evaluating delayed write in a multilevel caching file system. In *IFIP/IEEE Int'l Conf. on Dist'd Platforms*, pages 415–429, Feb. 1996.

[33] F. Olken. Efficient methods for calculating the success function of fixed space replacement policies. Technical Report LBL-12370, U.C. Berkeley EECS Dept. / Lawrence Berkeley Lab, May 1981.

[34] V. N. Padmanabhan and L. Qiu. The content and access dynamics of a busy Web server: Findings and implications. In *SIGCOMM*, Aug. 2000.

[35] N. T. Spring and D. Wetherall. A protocol-independent technique for eliminating redundant network traffic. In *SIGCOMM*, Aug. 2000.

[36] J. G. Thompson. Efficient analysis of caching systems. Technical Report UCB/CSD 87/374, U.C. Berkeley EECS, Oct. 1987.

[37] D. Wessels. Personal communication.

[38] A. Wolman, G. Voelker, N. Sharma, N. Cardwell, M. Brown, T. Landray, D. Pinnel, A. Karlin, and H. Levy. Organization-based analysis of Web-object sharing and caching. In *2nd USENIX Symposium on Internet Technologies and Systems*, Oct. 1999.

[39] A. Wolman, G. M. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. M. Levy. On the scale and performance of cooperative Web proxy caching. *Operating Systems Review*, 34(5):16–31, Dec. 1999.