Aliasing on the World Wide Web: Prevalence and Performance Implications[*]

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

Jeffrey Mogul
Compaq Western Research Lab
250 University Avenue
Palo Alto CA 94301 USA
JeffMogul@acm.org

ABSTRACT

Aliasing occurs in Web transactions when requests containing different URLs elicit replies containing identical data payloads. Conventional caches associate stored data with URLs and can therefore suffer redundant payload transfers due to aliasing and other causes. Existing research literature, however, says little about the prevalence of aliasing in user-initiated transactions, or about redundant payload transfers in conventional Web cache hierarchies.

This paper quantifies the extent of aliasing and the performance impact of URL-indexed cache management using a large client trace from WebTV Networks. Fewer than 5% of reply payloads are aliased (referenced via multiple URLs) but over 54% of successful transactions involve aliased payloads. Aliased payloads account for under 3.1% of the trace's "working set size" (sum of payload sizes) but over 36% of bytes transferred. For the WebTV workload, roughly 10% of payload transfers to browser caches and 23% of payload transfers to a shared proxy are redundant, assuming infinite-capacity conventional caches. Our analysis of a large proxy trace from Compaq Corporation yields similar results.

URL-indexed caching does not entirely explain the large number of redundant proxy-to-browser payload transfers previously reported in the WebTV system. We consider other possible causes of redundant transfers (e.g., reply metadata and browser cache management policies) and discuss a simple hop-by-hop protocol extension that completely eliminates all redundant transfers, regardless of cause.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Applications*; C.4 [**Performance of Systems**]: Measurement Techniques; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Client/server*

General Terms

Management, Measurement, Performance, Design

Keywords

World Wide Web, WWW, Hypertext Transfer Protocol, HTTP, aliasing, redundant transfers, duplicate suppression, Duplicate Transfer Detection, DTD, resource modification, caching, cache hierarchies, performance analysis, Zipf's law

[*] `$Id: aliasing.tex,v 1.68 2002/03/20 04:13:49 tpkelly Exp $`

1. INTRODUCTION

Aliasing occurs in Web transactions when different request URLs yield replies containing identical data payloads. Existing browsers and proxies perform cache lookups using URLs, and aliasing can cause redundant payload transfers when the reply payload that satisfies the current request has been previously received but is cached under a different URL. Awareness of this problem is slowly growing in the commercial world: A major cache appliance vendor now encourages site designers to make Web pages cache-friendly by avoiding aliasing [13, page 9]. Within well-administered sites, aliasing might decrease when such advice is heeded. Other trends, however, are moving the Web in the opposite direction and are beyond the control of individual sites. For example, commercial Web design tools typically include many small images, and a particular image bundled with such a tool is available through a different URL at each site that uses it.

Given the proliferation of technologies that create aliases and the potential for aliasing to cause redundant transfers, it is surprising that relatively little is known about the scale and consequences of this phenomenon. Only a few previous studies have considered the prevalence of aliasing in Web transactions, the performance penalty of conventional URL-indexed cache management in large multi-level cache hierarchies, or ways to eliminate redundant payload transfers [36, 50]. Few of the Web workload traces that researchers have collected can illuminate the relationship between request URLs and reply payloads, because they do not describe payloads in sufficient detail.

In this paper we quantify aliasing and the impact of URL-indexed cache management on browser and proxy cache miss rates by examining large anonymized client and proxy traces collected, respectively, at WebTV Networks in September 2000 and at Compaq Corporation in early 1999. The traces include message digests of reply payloads, allowing us to detect aliasing. We investigate whether conventional URL-indexed caching is primarily responsible for the high rate of redundant proxy-to-browser payload transfers previously reported in the WebTV system, consider other causes of redundant payload transfers, and discuss a simple way to eliminate all redundant transfers, regardless of cause.

1.1 Terminology

In this paper, we use several terms that require precise definition:
Payload: The data carried in the body of an HTTP response. We consider only full-body responses (status code 200); partial-body (status code 206) responses are extremely rare, and would complicate the discussion.
Payload hit: A cache reference that returns, from cache storage, exactly the payload the origin server would return at the time of ac-

cess, whether or not messages are exchanged with the origin server. A revalidation request followed by a "Not Modified" response is a payload hit, because it averts the payload transfer.

Payload miss: A reference for which the correct payload is not obtained from cache but rather from elsewhere (e.g., the origin server or from a remote intermediate cache).

Aliased payload: In the context of a given trace of Web transactions, a payload is aliased if it is accessed via more than one URL in the entire trace.

In our trace-driven simulations we do not consider caches that violate semantic transparency in the sense of RFC 2616 [23], i.e., we consider caches that either suffer payload misses or return exactly the same payload that the origin server would return at the moment of access.

Certain terms related to aliasing, such as "duplication," lack clear, widely-accepted definitions in the literature, and we will avoid them when discussing our own work.

2. RELATED WORK

While few studies have directly addressed our central topic, many have investigated aspects of the HTTP namespace and their impact on cache performance. In this section we review literature on the relationship between URLs and reply payloads.

2.1 Resource Modification

"Resource modification" is the complement of aliasing: requests containing identical URLs yield different reply bodies. Because it has direct implications for cache consistency and object "cachability," resource modification has been extensively studied. In Section 4 we compare the prevalence of aliasing and resource modification and report that more transactions are affected by the former.

Douglis et al. report that rates of resource modification in a trace from a corporate environment are high enough to substantially reduce the hit rates of conventional URL-indexed caches [20]. More recently, Brewington & Cybenko consider the burden that modification rates place on search engines [9]. After fitting a combination of exponential and Weibull models to their data, they report that roughly 10% of inter-modification intervals are 10 days or less and roughly 72% are 100 days or less. Brewington's doctoral thesis considers the problem of monitoring changing information resources in greater theoretical and empirical depth [8]. This research is based on polling URLs obtained from users who have requested notification when specified resources change, and might therefore reflect a sample of resources with atypical rates of change. Padmanabhan & Qiu analyze the dynamics of content creation, modification and deletion at the MSNBC Web site [43]. They report a median inter-modification interval of approximately 10,000 seconds and note that most alterations to files are relatively minor.

Resources expected to change frequently are often called "dynamic," although this poorly-defined term blurs the distinction between the process by which a response is generated, and whether it is "cachable." In practice, cache implementors and researchers employ heuristics to identify uncachable responses by looking either for signs of dynamic generation (such as "cgi" in a URL) or for metadata, such as cookies, implying that a resource gives a different response to every request. Wolman et al. report that Squid deems uncachable 40% of replies in a large trace collected at the University of Washington in May 1999; Zhang reports that customized and dynamic content together render roughly 7.1% of the objects in his trace uncachable [59].

Work by Wills & Mikhailov, however, casts doubt on the assumption that it is pointless to cache seemingly "dynamic" or "customized" content. They report that even if a previous access to a URL had returned a "Set-Cookie" header, in most cases the absence of a request cookie, or the presence of a different cookie, does not affect the reply payload returned for a subsequent access [54]. Repeated accesses to query resources at E-commerce sites sometimes return identical payloads [56]. Iyengar & Challenger exploited the cachability of dynamic replies at a large, busy Web server and report impressive performance gains [26]. Smith et al. report dynamic reply cache hit rates of 13.6% and 38.6% for two workloads [48].

Wolman et al. incorporate observed resource popularity and modification rates into an analytic model of hierarchical caching [58]. Their model illustrates the impact of resource modification rates on cache hit rates and suggests that cooperative caching schemes yield diminishing returns as client populations increase.

2.2 Mirroring

"Mirroring" typically refers to a special case of aliasing in which replicas of pages or entire sites are deliberately made available through different URLs. Shivakumar & Garcia-Molina investigate mirroring in a large crawler data set [47]. They report far more aliasing than appears in the WebTV client trace: 36% of reply bodies are accessible through more than one URL. Bharat et al. survey techniques for identifying mirrors on the Internet [6]. Bharat & Broder investigate mirroring in a large crawler data set and report that roughly 10% of popular hosts are mirrored to some extent [5].

Broder et al. consider approximate mirroring or "syntactic similarity" [10]. Although they introduce sophisticated measures of document similarity, they report that most "clusters" of similar documents in a large crawler data set contain only *identical* documents. In other words, simple aliasing is the dominant form of similarity in their workload.

2.3 Duplicate Suppression

Douglis et al. report that 18% of the full-body responses recorded at a corporate firewall that resulted in a new instance of a particular resource were identical to at least one other instance of a different resource [20].

Several "duplicate suppression" proposals address performance problems caused by duplication. The HTTP Distribution and Replication Protocol (DRP) employs payload digests to avoid unnecessary data transmission in deliberate replication over HTTP [52]. A DRP client obtains "index files" containing digests indicating the current state of resources, and the client can then request precisely those resources for which its copies are obsolete.

Mogul reviewed a variety of end-to-end duplicate-suppression schemes involving "hints" supplied by origin servers to clients, and by clients to caches. These proposals do not entirely eliminate the problem of redundant payload transfers, and a trace-driven simulation demonstrates that one such scheme yields 5.4% and 6.2% improvements in hit rates and byte hit rates, respectively. Even these modest gains are *upper bounds*, because they assume the full participation of all origin servers [35, 36].

Santos & Wetherall [46] and Spring & Wetherall [50] describe a general protocol-independent network-layer technique for eliminating redundant traffic by caching *packet* payloads and transmitting digests thereof to avoid redundant transfers. Muthitacharoen et al. designed a network file system for low-bandwidth environments that performs similar operations on chunks of files [40].

Inktomi's Traffic Server proxy cache product has included a technique called "content fingerprinting," which uses payload digests to avoid storing multiple copies of identical payloads [32]. Content fingerprinting suppresses duplicates in storage, but not on the network. Bahn et al. describe a similar scheme [3].

2.4 Data-Collection Methodology

In order to evaluate the performance impact of aliasing on large-scale browser-proxy cache hierarchies, we require a detailed record of (request, reply) transactions from a large client population. (Existing synthetic workload generators such as SURGE [4], WebPolygraph [53], and SPECweb [49] were not designed to mimic aliasing "in the wild," and are therefore inappropriate for our purposes.) The logs from a server might reflect large numbers of users, but they do not record all of the users' requests, and so are of little use in evaluating client or proxy caching.

In order to record transactions involving large numbers of both users and servers, researchers typically employ packet sniffers [21, 22] or proxy logs [19]. However, the use of caching proxies can complicate either approach. If a sniffer is located between a caching proxy and the Internet, it will fail to record requests served from the proxy cache. The logs of a caching proxy will not suffer from this problem, but such logs do not necessarily reflect the payloads that origin servers would provide: proxies might serve stale content unless they revalidate payloads with the origin server upon every cache hit. Moreover, a trace collected at a proxy normally fails to capture any user requests that hit in the client (browser) caches. Other problems with conventional proxy logs include inadequate detail, low-resolution timestamps, and poor clock synchronization in multiple-host proxy arrays [12, 15, 17, 28].

In principle, one could avoid such problems by collecting traces using an instrumented client; this could capture every user reference. Instrumented browsers have been used to collect traces from small user populations [14, 16]. It is difficult to instrument popular browsers today because source code is unavailable, but a client proxy such as Medusa [29] can collect much of the same data. The main problem with client-end data collection is the difficulty of deployment across a large client sample.

In collecting the anonymized traces we analyze, WebTV employed a *non-caching, cache-busting* proxy. A *cache-busting* proxy marks as uncachable all of the replies it sends to clients, effectively disabling browser caches. And because the proxy itself maintains no cache, it never suffers from the stale-response problem. Since the source code for the proxy was available, WebTV could instrument it to collect data not ordinarily logged (e.g., payload digests, which have on rare occasions been logged by modified proxies in the past [35, 36]). For more discussion of WebTV's trace collection methodology, see Reference [28].

A cache-busting proxy allowed WebTV to collect a very large trace from a large user population at reasonable cost. The WebTV trace is comparable in size to the largest traces used in HTTP namespace investigations. Unlike these proxy, sniffer, server, and crawler traces, however, the WebTV client trace can support trace-driven simulation of browser-proxy cache hierarchies.

2.5 Harmful Practices

The HTTP/1.1 specification is long and complex [23]. Not all servers are fully compliant, and the compliance of products does not always improve over time [30]. Non-compliance can clearly cause redundant payload transfers and other kinds of waste. However, redundant transfers can also occur if mechanisms introduced into HTTP/1.1 to improve cache correctness are used in strange but *compliant* ways. For instance, identical payloads served by a single site are sometimes accompanied by *different* entity tags [55], causing new-style "`If-None-Match`" revalidation attempts to fail where old-fashioned "`If-Modified-Since`" requests might succeed. In this case, the server is compliant with the specification, but not with the most efficient possible implementation.

	full trace	reduced trace
Clients	37,201	37,165
Server IP addresses	267,595	252,835
Server hostnames	536,451	412,509
URLs	40,756,045	32,541,361
Unique payloads	38,754,890	36,573,310
(URL, payload) pairs	54,910,572	44,785,808
Transactions	347,460,865	326,060,677
Bytes transferred		
Total		1,973,999,619,772
Unique payloads		639,563,546,204

Table 1: WebTV trace summary statistics.

Furthermore, several common practices that do not violate the protocol complicate the HTTP namespace in harmful ways. Mikhailov & Wills report, for instance, that content providers sometimes embed session identifiers in dynamically-written URLs rather than cookies [33]. Ad rotation often creates many minor variants of the HTML for a Web page, inflating resource modification rates. Padmanabhan & Qiu document other types of minor changes that occur frequently at the MSNBC site [43].

2.6 Summary

Existing literature touches on a number of issues surrounding aliasing on the Web, but the prevalence of this phenomenon across user-initiated transactions and the impact of URL-indexed cache organization on miss rates in multi-level cache hierarchies is poorly understood. Most proxy traces employed in empirical Web caching research shed no light on aliasing because they do not record data payloads or digests thereof. Data sets collected by Web crawlers often include payload digests but cannot support trace-driven simulations of cache hierarchies; they illuminate aliasing across *available* resources rather than *accessed* resources. The WebTV trace described in Section 3 is well suited to our investigation because it reflects all client requests and corresponding server replies in a large, cacheless production environment.

3. TRACES

We analyze aliasing in an anonymized Web client trace collected at WebTV Networks in September 2000, summarized in Table 1. The trace spans sixteen days and reflects over 347 million references to over 40 million resources by over 37,000 clients. It was collected using a specially instrumented "cache-busting proxy," as described in Section 2.4 and in Reference [28]. This allowed the proxy to record requests that would otherwise be served silently from browser caches.

In this paper we consider only successful (status code 200) transactions in the WebTV trace. We furthermore exclude transactions involving seventeen payloads for which accurate sizes are not available; these account for slightly over 100,000 transactions. Our reduced trace is summarized in the right-hand column of Table 1. Due to differences in trace-reduction procedures used in the two investigations, the summary statistics in Table 1 differ from those in Table 3 of Reference [28]. As in the earlier paper we associate with each payload a single size that includes protocol overhead (HTTP headers). In the present study we add to each payload's Content-Length a median header size of 247 bytes, whereas the earlier paper used the maximum size-related field observed in any transaction involving a payload.

WebTV clients are inexpensive devices that enable Web surfing on conventional television sets. Most are used in homes, and all

Clients	at least 21,806
Server hostnames	at most 454,424
URLs	19,644,961
Unique payloads	30,591,044
(URL, payload) pairs	34,848,044
Transactions	78,913,349
Bytes transferred	
Total	902,792,408,397
Unique payloads	537,460,558,056

Table 2: Compaq reduced trace summary statistics.

rely on telephone modems operating at roughly 33.6 Kbps. For our purposes the most important features of the WebTV trace are:

- it reflects activity in a *cacheless* system, i.e., both browser and proxy caches were disabled during data collection;
- the clients never used HTTP features such as Ranges or Delta encoding [38], so every reply body is self-contained; and
- it contains anonymized message digests of every reply body.

The last feature is crucial to the present investigation, for it illuminates the relationship between a URL and the data payload returned as a result of a specific access to that URL at a given instant. The first feature ensures that the payloads recorded in every transaction are those returned directly from the origin server; there is no chance that the WebTV proxy served (and logged) a stale payload from its cache. Another nice feature of the WebTV trace is that it contains no “robots,” which can only be identified with complex and unreliable heuristics [1].

While data collected via polling or crawling would shed light on aliasing across *available* resources, we must focus on a *request stream* in order to evaluate the cache performance impact of aliasing. Furthermore a *client* trace is essential in order to evaluate the impact of aliasing on browser/proxy cache *hierarchies*. The WebTV client trace is particularly attractive because it reflects workload in a well-integrated production environment controlled by a single organization; performance enhancements suggested by workload analysis are far easier to implement in such environments than in the overall Web.

The WebTV system is a somewhat peculiar environment, and it is reasonable to suspect that thin-client surfing might differ systematically from browsing with conventional rich clients. We therefore repeated our performance evaluations using a large proxy trace recorded on the Compaq corporate network. This trace, described in detail in Reference [35], is summarized in Table 2.[1] Like the WebTV trace, the Compaq trace contains payload digests, but because browser caches were enabled it reflects only browser cache misses. Therefore the Compaq trace cannot be used to evaluate browser cache performance. As with the WebTV data, we use a reduced Compaq trace containing only status-200 transactions; a small number of erroneous transactions are also excluded.

3.1 Trace Characteristics

The WebTV trace is large not merely in terms of number of clients and transactions but also in terms of its “working set.” The sum of distinct payload sizes in the WebTV trace is roughly 600 GB. At the time the WebTV trace was collected, thirty-one production-grade cache products competed in a “Cache-Off” benchmark exercise [45]. The mean capacity of these caches was 83.5 GB and the median size was 42 GB. The WebTV workload could fill the largest entrant’s cache (315 GB) nearly twice. However the trace’s working set is not impossibly large by the standards of September 2000; the bank of modified WebTV proxies that collected the trace had a total capacity of roughly 600 GB. Similarly, the sum of distinct payload sizes requested by typical clients in the trace is moderately large for a set-top device, but not excessively so. The median client receives under 20 MB of distinct payloads, and over 26% of the client devices that generated the WebTV trace had larger browser caches [28]. In Section 5 we consider browser and proxy caches sufficiently large that they suffer no capacity misses.

[1]The number of client hosts in Table 2 is an underestimate, due to lost backup tapes.

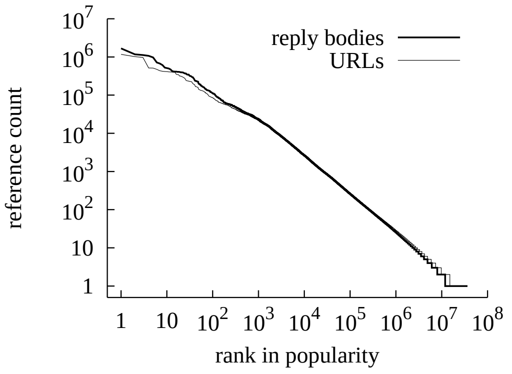


Figure 1: Zipf-like reference counts of URLs and reply bodies.

	URLs	Unique payloads
Count	32,541,361	36,573,310
Zipf α	1.0341 ± 0.000032	0.9376 ± 0.000043
Zipf β	7.6313 ± 0.000227	6.9112 ± 0.000308
R^2	0.969766	0.928005

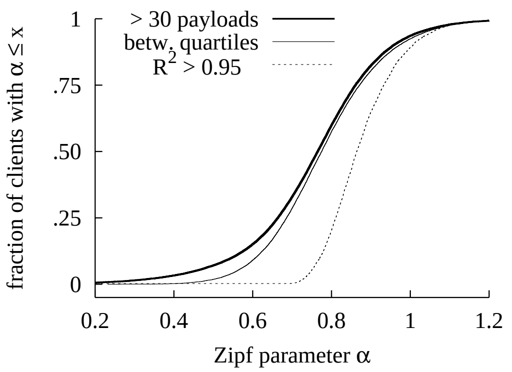
Table 3: WebTV trace Zipf parameters.

The popularity distributions of URLs and reply bodies affect cache performance, and these are shown in Figure 1. As in most Web workloads studied to date the popularity distribution of URLs is Zipf-like, as is that of distinct data payloads. Table 3 reports Zipf parameters for the popularity distribution of both URLs and reply bodies, obtained by fitting to the WebTV data linear least-squares models of the form

$$\log_{10}(\text{reference count}) = -\alpha \log_{10}(\text{popularity rank}) + \beta$$

using the algorithms described in References [27, 44]. The Zipf α parameters in the WebTV client trace are remarkably close to unity. Cunha et al. report similar findings on the Boston University client trace [16]. By contrast, the Zipf parameter is often higher at servers [43] and lower at proxies [7]. For an interesting analysis of why the Zipf α varies at different levels in a cache hierarchy, see Padmanabhan & Qiu [43]. Breslau et al. discuss the implications of Zipf-like popularity distributions for caching [7].

We furthermore compute the Zipf α parameter of the payload popularity distribution for *each client* in the WebTV trace. Figure 2 shows the distributions of α across three subsets of the client population: 1) all clients that request more than thirty distinct payloads (this excludes only a handful of clients), 2) clients whose number of distinct referenced payloads is between the 25th and 75th percentiles, and 3) clients whose model fit is particularly good ($R^2 > 0.95$). The figure displays separately the distributions of R^2 for the first two groups. Three remarkable features are apparent in Figure 2: For most clients a Zipf model describes the popularity of accessed payloads reasonably well ($R^2 > 0.9$). Further-



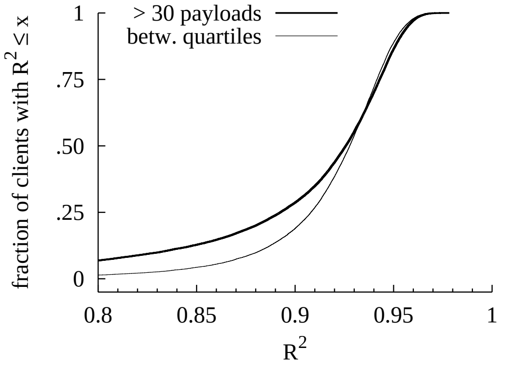


Figure 2: Top: CDF of Zipf α across client sub-populations. Bottom: CDF of R^2.

more, whereas α for the overall trace (shared proxy serving cacheless clients) is roughly 0.938, it is *lower* in over 87% of individual client reference streams. At present we have no theoretical explanation for why α should be higher at a shared proxy than at clients. Finally, α is noticeably higher in clients for which the Zipf model fit is close.

Figure 3 shows the cumulative concentration of references across URLs and payloads sorted in descending order of popularity. The top one percent of payloads account for over two thirds of all transactions and the top ten percent account for nearly 85%; for URLs the figures are respectively 62.2% and 82.4%. Concentration of references in the WebTV client trace is much stronger than in proxy traces (e.g., Figure 5a of Arlitt et al. [2]). Browser caches filter reference concentration, as well as reference locality, from the original client reference streams, so that both locality and concentration are markedly lower in the reference stream that reaches proxies. See

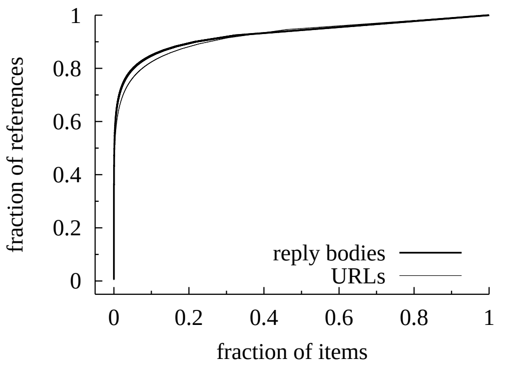


Figure 3: Concentration of references.

Trace set	Response content-length mean	Response content-length median
WebTV (includes HTTP headers)	6,054	1,821
Compaq	11,192	2,894
Berkeley Home IP [25]	7,964	2,239
ATT-delta [37]	7,881	3,210
HP cable modem [2]	21,568	4,346
Washington [58]	7.7KB	not avail.

Table 4: Response sizes in various traces.

MIME type	Transactions % by count	Transactions % by bytes	Payloads % by count	Payloads % by bytes
image/gif	68.389	34.391	17.247	5.727
image/jp[e]g	18.627	25.843	24.854	18.441
text/html	10.255	22.391	54.212	43.902
app'n/[x-]javascript	1.169	0.625	1.142	0.088
audio/(midi,x-midi,mid)	0.253	1.171	0.089	0.159
video/mpeg	0.077	9.808	0.291	20.843
app'n/octet-stream	0.034	0.716	0.038	1.390
video/quicktime	0.002	0.400	0.010	1.143
video/x-msvideo	0.001	0.547	0.009	1.440
all other	1.192	4.106	2.107	6.867

Table 5: MIME type distribution of WebTV trace.

Figure 10 of Reference [43] for *server* vs. proxy reference concentration.

3.2 Trace Representativeness

Any study that generalizes from one or two traces must consider whether they are representative of Web use in general. We compared our data sets with traces used in prior literature in terms of Zipf parameters (Section 3.1), response sizes, and MIME type distributions.

Table 4 shows mean and median response body sizes from several relatively large, recent trace sets. Our WebTV and Compaq traces roughly span the range of means and medians, except for those of the HP cable modem trace. The WebTV sizes are similar to (but slightly smaller than) the Berkeley sizes, consistent with the use of slow final hops in both environments. The Compaq sizes are not inconsistent with those from the AT&T and Washington broadband environments. We do not know why the cable modem sizes are so large.

Table 5 shows the fraction of transactions, bytes transferred, payloads, and working set associated with popular MIME types; all types that account for 1% or more in any category are shown. In terms of transactions and bytes transferred, the WebTV trace is roughly similar to other workloads reported in the literature, e.g., the AT&T trace described in Table 1 of Douglis et al. [20]. JPEG files are more prominent in WebTV's client trace, probably because client caches handled many JPEG accesses in the AT&T trace. In terms of distinct payloads, HTML is far more prevalent in the WebTV trace (54% vs. 24%). The practice of decomposing logical pages into multiple HTML frames, more common in September 2000 than in November 1996, might partly explain the difference.

Wolman et al. collected a large Web trace at the University of Washington using a packet sniffer in May 1999. Figure 1 of Reference [57] reports the distribution of MIME types in this trace. Image files account for more transactions and more bytes trans-

URLs	32,541,361
Modified URLs	1,859,929
Unique payloads	36,573,310
Aliased payloads	1,821,182
(URL, payload) pairs	44,785,808
Transactions	326,060,677
w/ modified URLs	32,277,753
w/ aliased payloads	176,595,754
Payload sizes	
Range (min–max)	40–91,397,479
Median	5,487
Mean	17,487
Sum	639,563,546,204
Sum of aliased	19,726,808,472
Transfer sizes	
Median	1,821
Mean	6,054
Sum	1,973,999,619,772
Sum of aliased	711,717,843,218

Table 6: WebTV reduced trace aliasing statistics.

ferred in the WebTV trace, probably due to client caching on the University of Washington campus.

On the basis of the available quantitative evidence, therefore, the WebTV and Compaq traces used in this paper appear consistent with other recent traces.

4. PREVALENCE OF ALIASING

For the purposes of this section, a transaction record is a pair (U,P) where U is a request URL and P is a reply data payload. We say that a reply payload P is *aliased* if there exist two or more records $(U,P),(U',P)$ containing the same reply payload P but different URLs U and U'. Similarly, we say that a URL U is *modified* if there exist two or more transactions containing U as the URL and different reply payloads P and P'. The *degree* of a payload is the number of distinct URLs that appear with it in transaction records, and the degree of a URL is the number of distinct reply payloads that appear with it in the trace. Aliased payloads and modified URLs each have degree two or greater.

Table 6 shows that aliased payloads account for over 54% of transactions and 36% of bytes transferred in the WebTV trace, suggesting that conventional URL-indexed caching might lead to many redundant transfers and much redundant network traffic. We address these issues in Section 5. The table also shows that under 5% of payloads are aliased, and aliased payloads constitute only 3% of the working set.

Note that because the WebTV trace was made with all caching disabled, many of the aliased-payload transactions in Table 6 would not have been seen in a more typical environment; they would have been avoided by traditional client caches. However, caching would also reduce the total number of transactions, so the fraction of aliased payloads in a cache-filtered reference stream depends (in part) on whether aliased payloads experience more or less locality than others, as well as on the specific cache configuration. We do not investigate this issue.

The distributions of the degrees of payloads and URLs in the WebTV trace are shown on the left in Figure 4. Fewer than 5% of payloads are aliased, but one is accessed via 348,491 different URLs. Similarly only 5.7% of URLs are modified, but one yields 491,322 distinct payloads. This analysis downplays the prevalence of aliasing and modification because it does not consider the number of times that different (URL, payload) pairs occur in the trace. The plot in the center shows the distributions of payload and URL degrees weighted by reference count. Whereas only 10% of transactions involve modified URLs, over 54% involve aliased payloads; *reply-payload aliasing affects more transactions than resource modification.* The plot on the right shows the distribution of bytes transferred by the degree of the payload involved; 36% of the bytes transferred in the WebTV workload involve aliased payloads.

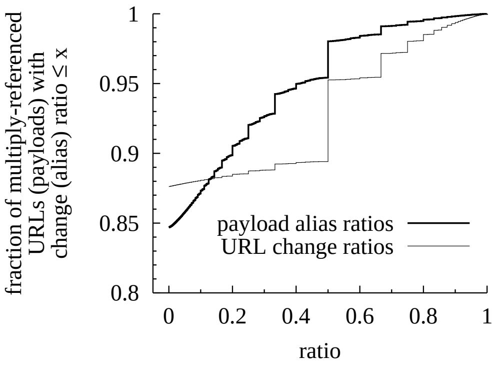


Figure 5: CDFs of change and alias ratios.

In over 41 million successful transactions (12.72%) a payload is accessed through a different URL than in the previous access to the same payload. By contrast, under 14.3 million transactions (4.37%) involve a different payload than the previous transaction with the same URL. Here again the prevalence of aliasing exceeds that of resource modification. Note that this does not imply that aliasing causes more cache misses than resource modification; in fact, the reverse might be true. Our simulations do not address this question.

Following Douglis et al. [20] we compute for each multiply-referenced URL its "change ratio," the fraction of its accesses that return a different data payload than its previous access. We furthermore compute for each multiply-referenced *payload* an analogous metric, the "alias ratio," defined as the fraction of its accesses made through a different URL than its previous access. The distributions of change ratios and alias ratios across multiply-referenced URLs and payloads, respectively, are shown in Figure 5. The figure shows that 15.3% of multiply-referenced payloads are aliased and 12.4% of multiply-referenced URLs are modified. However the figure also shows that alias ratios are generally lower than change ratios. For example, only 2% of multiply-referenced payloads have alias ratios above 0.5 whereas 4.7% of multiply-referenced URLs have change ratios over 0.5.

4.1 Aliasing and Response Attributes

Techniques meant to eliminate redundant transfers usually impose some costs. If we could impose those costs only on those subsets of responses that are most likely to benefit from an alias elimination technique, we could (in principle) reduce overall costs without similarly reducing overall benefits.

Table 7 shows the prevalence of aliasing among popular MIME types in the WebTV trace. The table uses the same sort order as Table 5. Roughly 4.3% of payloads are served (at different times) with more than one MIME type; in such cases we define the payload's type to be the most common type.

Aliasing is most common among MIDI payloads: 35% of MIDI payloads are accessed via two or more different URLs, and over 80% of MIDI transactions involve aliased payloads. However Ta-

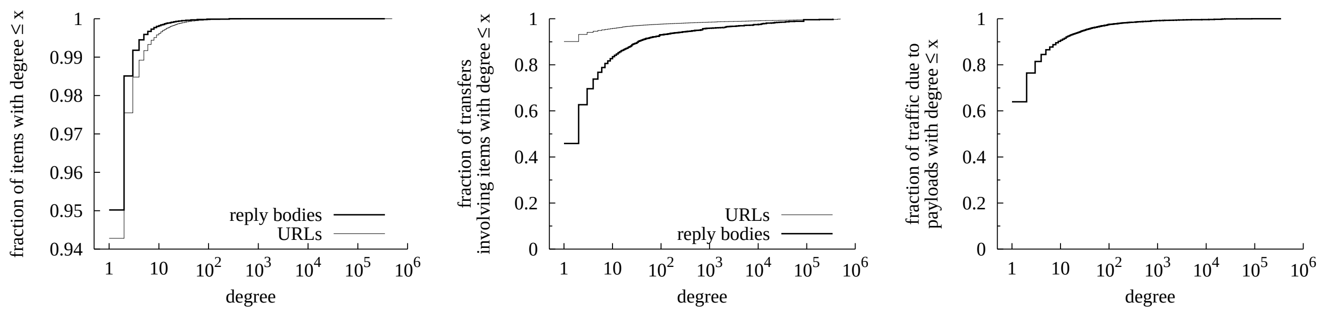


Figure 4: Left: CDF of payload and URL degrees. Center: CDF of transactions by degree of URL & payload involved. Right: CDF of bytes transferred by degree of payload involved.

	Transactions w/ Aliased Payloads		Aliased Payloads	
MIME type	% by count	% by bytes	% by count	% by bytes
image/gif	66.113	62.608	13.016	11.901
image/jp[e]g	30.655	30.748	6.976	6.472
text/html	15.993	12.729	1.577	1.172
app'n/[x-]javascript	66.564	67.410	1.863	4.370
audio/(midi,x-midi,mid)	82.854	81.833	35.157	32.052
video/mpeg	32.472	13.284	6.422	1.932
app'n/octet-stream	63.557	19.263	10.563	3.886
video/quicktime	7.583	1.205	1.515	0.488
video/x-msvideo	8.882	5.574	2.125	1.472
all other	47.089	21.200	3.324	2.275

Table 7: Prevalence of aliasing by MIME type in WebTV trace.

ble 5 shows that MIDI accounts for under 2% of all traffic and under 1% of all transactions.

GIF files account for over two thirds of transactions and over one third of bytes transferred in the WebTV trace (Table 5), and roughly two thirds of GIF transactions involve aliased payloads (Table 7). Taken together, these facts imply that *nearly half of all transactions involve aliased GIF payloads* ($0.66113 \times 0.68389 = 0.45214$). By contrast, aliasing is far less prevalent among HTML and JPEG payloads, which together account for roughly 29% of transactions and 48% of bytes transferred; fewer than 7.5% of transactions involve aliased HTML or JPEG payloads. Our findings are consistent with the hypothesis that Web authoring tools account for much of the aliasing in Web transactions; unfortunately our traces are anonymized in such a way as to prevent more detailed investigation of the issue.

Techniques that attempt to eliminate redundant payload transfers might be best applied to MIME types, such as images and audio, subject to frequent aliasing. Frequently-used types that seldom suffer aliasing, such as HTML, should perhaps not be burdened with additional overheads.

We also examined the relationship between aliasing and payload size. Figure 6 shows several distributions involving the sizes of payloads in the WebTV trace. The top row of distributions shows that aliased payloads, and the transactions and bytes transferred due to them, tend to be smaller than their non-aliased counterparts. However when we examine particular MIME types this generalization does not always hold. For example, aliasing is associated with slightly larger payload sizes in JPEG transactions and HTML traffic. Techniques that attempt to eliminate redundant payload transfers should add a minimal number of header bytes, since the bias towards aliasing of small payloads implies that potential benefits can easily be squandered.

1	Mirrored content	`http://mir1.bar.com/img.gif` `http://mir2.bar.com/img.gif`
2	Within-site	`http://bar.com/image.gif` `http://bar.com/i.gif`
3	Different sites, same `abs_path`	`http://bar.com/img.gif` `http://foo.com/img.gif`
4	Everything different	`http://bar.com/image.gif` `http://foo.com/i.gif`

Table 8: Causes of aliasing.

4.2 Causes of Aliasing

Aliasing can arise in several different ways, e.g., deliberate mirroring, aliasing within a single site, and identical content available at different sites. We can further decompose the last cause into cases where the `abs_path` component of the URL is the same, or different. Table 8 provides examples of the possibilities.

Knowing the cause of aliasing can help us decide where to focus efforts at remediation. A site can replace an ad-hoc mirroring strategy with a CDN, which does not introduce aliasing into the HTTP namespace [18]. Site administrators can avoid type 2 aliasing, following the advice of a CacheFlow white paper on cache-friendly site design [13]. Unfortunately the widespread use of Web authoring tools can cause type 3 aliasing, and this is beyond the control of individual sites. Furthermore aliasing occurs even *within* such tools: DreamWeaver [31], for instance, contains 632 unique image files under *642* different filenames.

The raw WebTV trace is anonymized in such a way that aliasing of types 1 and 3 cannot be distinguished. Furthermore the reduced trace that we use in our empirical work omits anonymized `abs_path` fields, preventing us from distinguishing between types 3 and 4. We can, however, identify cases where different URLs contain identical versus different host components.

Payloads of degree 2, i.e., payloads accessed via exactly two different URLs, fall into exactly one of the categories in Table 8. Degree-2 payloads account for 70% of aliased payloads, 31% of transactions involving aliased payloads, and 34.6% of aliased payload bytes transferred. 80.56% of degree-2 payloads are accessed via URLs with different host components; the remainder are cases of within-site aliasing.

5. PERFORMANCE IMPLICATIONS

In any reference sequence, the first access to a given payload cannot be served from cache; we refer to these as "new-payload"

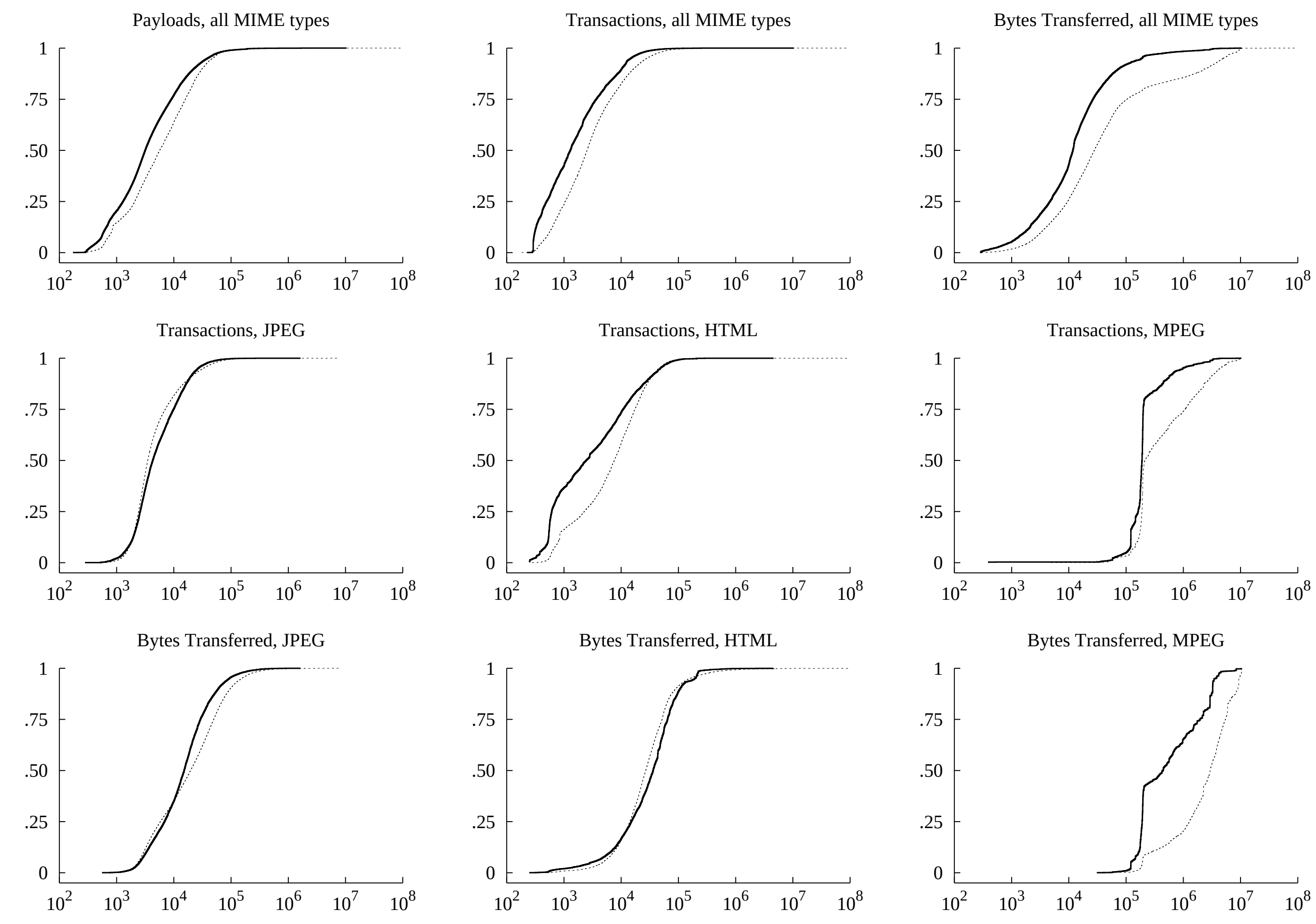


Figure 6: CDFs by payload size for all payloads (top row) and three popular MIME types. Solid lines indicate aliased payloads, transactions involving aliased payloads, and aliased bytes transferred; dashed lines non-aliased. All horizontal scales are identical and show payload size in bytes.

misses. In this section we compare the payload miss rates of conventional URL-indexed caches with new-payload miss rates. The use of URLs to organize and locate payloads in conventional caches entirely accounts for the difference between the two, and as we explain below, new-payload miss rates represent an *achievable* bound on the performance of a sufficiently large cache.

As explained in Section 1.1 we use the terms “payload hit” and “payload miss” to mean respectively “required payload is obtained from cache” and “payload must be fetched from elsewhere.” In this section we assume infinite-capacity caches that require no removal policy and suffer no capacity misses. We furthermore assume that caches are semantically transparent, i.e., that a cache hit supplies exactly the same payload as the origin server would. This can easily be achieved in practice by revalidating every request, but for brevity we do not discuss freshness checks and revalidations in this section.

5.1 Abstract Cache Models

A conventional “URL-indexed” cache stores the most recently received data payload in association with each requested URL. In addition to new-payload misses, an infinite-capacity URL-indexed cache suffers a payload miss, and resulting redundant transfer, if it has previously received the payload that satisfies the current request, but this payload is not currently cached in association with the current request URL.

Just as it is possible to design a cache that wastes no storage on multiple copies of payloads [3, 32], it is possible to design a cache that suffers *only* from new-payload misses (assuming infinite capacity). One can construct such a “frugal” cache by assuming that the cache computes a message digest of every stored payload, that the cache maintains an index mapping digest values to stored payloads, and that every payload-bearing response message received by the cache is preceded by a digest of that message’s payload, computed by the message sender (i.e., proxy or origin server). This allows the cache to avoid the payload transfer if it already stores a copy of the payload. The strategy is very simple: “1) cache forever every payload you receive, and 2) before receiving a payload, verify via digest lookup that you don’t already have it.” This approach ensures that the current request is served from cache if the required payload has ever been received before, and therefore only new-payload misses occur; redundant transfers cannot.

Section 7 sketches a realizable protocol design that can avoid redundant payload transfers in this manner. For the present we simply note that new-payload miss rates represent an *achievable* bound on the performance of infinite-capacity caches. Figure 7 describes, in pseudocode, how our two cache models process requests. A “frugal” cache conceptually maintains *two* caches of received payloads: a “u_cache” indexed by URLs and a “d_cache” indexed by payload digest. As noted above, we omit consistency checks from our pseu-

Conventional URL-indexed cache

```
if cache[URL] == correct payload
    conventional_payload_hit++
else
    new_payload_miss_or_redundant_transfer++
    send URL
    receive payload
    cache[URL] := payload
```

"Frugal" cache

```
if u_cache[URL] == correct payload
    conventional_payload_hit++
else
    send URL
    receive payload digest
    if d_cache[digest] == correct payload
        redundant_transfer_avoided_hit++
        send "don't bother"
    else
        new_payload_miss++
        send "proceed"
        receive payload
        d_cache[digest] := payload
        u_cache[URL] := payload
```

Figure 7: URL-indexed and "frugal" caches.

docode for brevity and clarity; such checks are necessary in order to ensure semantic transparency.

Our idealized cache models differ only in their susceptibility to payload misses due to the way they associate stored payloads with URLs. In other words, URL-indexed caching accounts for the difference between the payload miss rate of an infinite-capacity conventional cache and the new-payload miss rate inherent in the reference sequence. By comparing the two we can quantify precisely what fraction of URL-indexed cache payload transfers are redundant.

Aliased payloads can cause redundant transfers for URL-indexed caches, but are not the only cause. For example, a single resource (URL) whose value alternates between two payloads can also cause redundant transfers. Multiple causes may occur together, so a redundant transfer could fall into several categories. The abstract models in Figure 7, and our simulations in Section 5.2, do not isolate the contribution of aliasing to redundant transfers.

5.2 Simulation Results

We computed new-payload and URL-indexed payload miss rates and byte-weighted payload miss rates for 1) the aggregate browser cache population of over 37,000 clients, 2) a shared proxy cache serving cacheless clients, 3) a proxy serving infinite URL-indexed browser caches, and 4) a proxy serving infinite "frugal" browser caches.[2] Table 9 shows our results, including the percentage of redundant payload retrievals made by an infinite URL-indexed cache. The table separately reports cold and warm proxy simulation results; we used the first nine days of transactions in the sixteen-day WebTV trace to warm the simulated proxy for the latter. We do not report warm client results because at no time are all client caches equally warm: At any given point in the overall trace, some clients have issued many requests while others have issued few or none.

[2]Our simulations ignore "`no-store`" Cache-Control directives, which forbid payloads from being cached ("`no-cache`" merely requires that the payload be re-validated on every access). Only 0.14% of WebTV replies carry `no-store`; had our simulated caches heeded these directives the impact on payload miss rates would have been negligible.

Our results show that conventional URL-indexed caching entails large numbers of redundant transfers at both levels of the cache hierarchy: nearly 10% of payload transfers to clients and over 20% of payload transfers to a shared proxy are redundant. Even if redundant proxy-to-browser payload transfers are eliminated by "frugal" browser caches, nearly 12% of payload transfers to a URL-indexed proxy would be redundant. Under 4% of the network traffic between the proxy and infinite-capacity conventional clients is redundant, but over 13% of the traffic reaching a URL-indexed proxy serving URL-indexed or cacheless clients is redundant.

The last two rows of Table 9 show simulated payload miss rates and byte-weighted payload miss rates for an infinite-capacity shared proxy serving the Compaq workload. For the warm-proxy results we warm the simulated cache with the first 50 million transactions and tabulate payload miss rates based only on subsequent transactions. The Compaq results are roughly similar to the WebTV results: Over 17% of a URL-indexed cache's payload retrievals are redundant, as is roughly 12% of origin-to-proxy traffic.

We have not yet extended our simulator to model finite caches of various sizes. We expect, however, that the finite-cache results would lie between our infinite-cache and cacheless results.

6. EXPLAINING REDUNDANT TRANSFERS

The original motive for our investigation of aliasing was to explain the high rates of redundant proxy-to-browser payload transfers previously reported in the WebTV system [28]. Actual client payload miss rates are far higher than predicted by a simulated client-cache model that included only new-payload misses and capacity payload misses.

Redundant transfers can result from at least three causes: 1) faulty metadata supplied by origin servers or intermediaries, 2) inappropriate browser cache management, and 3) URL-indexed cache organization. It now appears that URL-indexed caching accounts for a substantial fraction of redundant payload transfers to WebTV clients, but not all of them. A thorough investigation of the remaining possibilities is the subject of our ongoing research; we offer a few tentative observations below.

Inappropriate metadata appears frequently in reply headers, and sometimes takes surprising forms. For instance, in the WebTV trace, different replies from the same server containing the same payload sometimes contain *different* entity tags. This curious phenomenon can cause "`If-None-Match`" revalidation attempts to fail needlessly, resulting in redundant payload transfers. Other researchers have explained this problem, which arises when large server farms fail to harmonize entity tags across server replicas [55].

Mogul investigated erroneous HTTP timestamps in a large trace and reported that 38% of responses contained impossible `Date` header values, and 0.3% had impossible `Last-Modified` values [34]. Some timestamp errors might cause transparency failures; others might cause needless revalidations. Wills & Mikhailov report a different kind of timestamp error: the `Last-Modified` reply header of a resource sometimes changes even when the reply body does not [55].

Anecdotal evidence suggests that Web design tools do not encourage content creators to associate reasonable expiration dates with pages. This is unfortunate because many commercial sites might incorporate business rules into `Expires` headers, e.g., "resources are only modified during business hours"; however, such

Simulated cache	Payload Miss Rates URL-indexed	new-payload	% redundant	Byte-Weighted Payload Miss Rates URL-indexed	new-payload	% redundant
cold ∞-cache clients	29.45	26.57	9.78	54.02	52.00	3.75
cold proxy serving cacheless clients	14.35	11.22	21.85	37.37	32.40	13.31
∞ URL-indexed clients	48.55	38.08	21.55	69.02	59.97	13.11
∞ frugal clients	47.83	42.21	11.75	69.27	62.30	10.06
warm proxy serving cacheless clients	12.93	9.93	23.14	35.58	30.40	14.55
∞ URL-indexed clients	46.30	35.74	22.80	67.42	57.77	14.32
∞ frugal clients	45.48	40.09	11.85	67.79	60.24	11.14
cold Compaq proxy, caching clients	46.84	38.77	17.24	67.49	59.53	11.79
warm Compaq proxy, caching clients	44.90	36.58	18.54	65.50	56.56	13.65

Table 9: URL-indexed and new-payload miss rates and % of URL-indexed payload transfers that are redundant.

practices seem to be rare. Finally, origin servers are not the only source of faulty metadata: for example, the popular Squid proxy does not update cached object headers after revalidations [42].

The WebTV browser cache might go too far in its efforts to avoid serving stale content to users. It truncates expiration dates to a maximum of 24 hours, and it *evicts* expired items rather than re-validating them with conditional GET requests [11, 51]. The Mozilla browser cache, by contrast, is designed to comply with the letter and spirit of HTTP/1.1 [24, 39]. WebTV's strategy prevents transparency failures when expiration dates are overly optimistic and might simplify implementation in memory-constrained client devices, but it might also inflate client miss rates. In future work we intend to quantify the relative contributions of faulty metadata and browser caching policies to redundant transfers.

7. AVOIDING REDUNDANT TRANSFERS

In Section 5.1, we described an abstract model for a cache that suffers only new-payload misses. Here we sketch how this could be realized in a practical protocol design, as an extension to HTTP. Our design averts *all* redundant payload transfers, including but not limited to those caused by aliasing. We call our design "Duplicate Transfer Detection" (DTD). DTD can be applied both to client and proxy caches.

First, consider the behavior of a traditional HTTP cache. Such a cache is URL-indexed: if the cache finds that it does not currently hold an entry for a requested URL U, this is a cache miss. On a miss, the cache issues or forwards a request for the URL towards the origin server, which would normally send a response containing payload P. If the cache holds an expired entry for U, it may send a "conditional" request, and if the server's view of the resource has not changed, it may return a "Not Modified" response without a payload. Real HTTP caches differ from the abstract URL-indexed model defined in Section 5.1 because they implement HTTP's cache-consistency mechanisms, and so may suffer redundant transfers resulting from inappropriate metadata (see Section 6) as well as from aliasing.

Now consider an idealized, infinite cache that retains in storage every payload it has ever received, even those that a traditional HTTP cache would not treat as valid cache entries. A finite, URL-indexed cache differs from this idealization because it implements both an update policy (it only stores the most recent payload received for any given URL), and a replacement policy (it only stores a finite set of entries, selected for maximum expected value).

The concept behind Duplicate Transfer Detection is quite simple: If our idealized cache can determine, before receiving the payload, whether it had ever previously received P, then we can avoid transferring that payload. Such a cache would experience only new-payload misses and would never suffer redundant payload transfers. A finite-cache realization of DTD would, of course, also suffer capacity misses.

How does the cache know whether it has received a payload P before the server sends the entire response? DTD follows the model of the abstract "frugal" cache described in Section 5.1. The cache maintains one set of cache entries but two lookup tables: one indexed by URL, and one indexed by the digest of each stored payload. If a DTD cache finds no fresh entry under the requested URL U, it forwards a (possibly conditional) request to the origin server. If the server replies with a payload, it first sends the digest D of the payload, and the cache checks for a stored entry with a matching digest value. Upon a digest match, the cache can signal the server not to send the payload (although the server must still send the HTTP message headers, which might be different). Thus, while DTD does not avoid transferring the request and response message headers, it can avoid any redundant payload transfer. We say that a "DTD hit" occurs when DTD prevents a payload transfer that would have occurred in a conventional URL-indexed cache.

An idealized, infinite DTD cache stores *all* payloads that it has received. In particular, it does not delete a payload P from storage simply because it has received a different payload P' for the same URL U. A realistic, finite DTD cache will eventually delete payloads from its storage, based on some replacement policy. A DTD cache might benefit from retaining old cache entries that other cache replacement and update algorithms would discard, speculating that such an entry will yield a future DTD hit.

7.1 Practical Issues for DTD

Practical implementation of DTD requires the solution of several problems. We mention a few of the problems here, but defer their solutions, and discussion of how to define DTD as a simple, compatible extension to HTTP, to a future paper.

When a client detects a DTD hit, how does it avoid the payload transfer? In one approach, the server sends the response headers (including the digest) but defers sending the payload until the client sends an explicit "proceed" request. In an alternative approach, the server sends the payload immediately, but stops if the client sends an "abort" message. (Note that HTTP would have to be extended to support either of these mechanisms.) The "proceed" model imposes an extra round-trip time (RTT) on every new-payload or capacity miss, but never sends any aliased payload bytes. (A more intricate form of the "proceed" model could amortize this delay over several misses.) The "abort" model does not impose additional delays, but the abort message may fail to reach the server in time to do any good. Choosing the right tradeoff will require experimental work.

The choice of digest algorithm also requires some tradeoffs. The algorithm must resist accidental or malicious collisions, but it must also not be expensive to compute, nor should the digest representation consume too many header bytes. A cryptographic hash algorithm such as SHA-1 [41] might have the right properties.

7.2 Similar Proposals

DTD is an application-level analogue of the router-level approach proposed by Santos & Wetherall [46] and Spring & Wetherall [50]. The two approaches are in some sense complementary, because each can avoid some redundant data transfers eliminated by the other. For example, the router-based approach can eliminate transfers of common prefixes of slightly different payloads, while DTD does not suffer from the re-packetization potentially caused by pipelining in HTTP. The approaches also differ in adoption dynamics: the router-level technique is easier to deploy for an organization that controls network infrastructure, while the application-level technique may be preferable for a single organization that controls two levels of a cache hierarchy (e.g., the client and proxy caches of AOL or WebTV).

8. SUMMARY

Our analysis of a large, detailed, recent Web client trace reveals that aliasing occurs frequently in Web transactions. Conventional URL-indexed caches are susceptible to misses caused by aliasing, and in our simulations such caches generate many redundant payload transfers: roughly 10% of payload transfers to infinite-capacity browser caches and over 20% of payload transfers to an infinite shared proxy serving either cacheless or infinite clients are redundant.

Aliasing is not the only cause of redundant payload transfers, so we describe a simple and completely general way to eliminate all redundant transfers, regardless of cause: Duplicate Transfer Detection. In future work we intend to quantify precisely the relative contributions of aliasing, faulty metadata, and inefficient cache management to the problem of redundant payload transfers, and describe in greater detail the latency and bandwidth savings achievable with DTD.

9. ACKNOWLEDGMENTS

We thank WebTV Networks for allowing us to study their trace, and especially Stuart Ozer, Arnold de Leon, and Jay Logue (and others listed in [28]) for gathering the trace, and David Surovell and Jake Brutlag for describing how WebTV client caches work. Compaq's Western Research Lab gave us access to their trace and generous equipment support, and Geoff Voelker of UC San Diego lent us additional system time. Glenn Cooper (Compaq WRL) and Jeff Kopmanis (U. Michigan AI Lab) provided computer support services.

Mikhail Mikhailov explained several puzzling phenomena we observed in our data, and with Brian Davison and Dave Pennock directed us to relevant literature. Mikhailov furthermore performed for us several tabulations of his own data sets to investigate reply header sizes and cookie sizes. Martin Arlitt of HP Labs answered several statistical questions. Mark Manasse helped us apply the Birthday Paradox. Peter Honeyman guided Kelly's research at the University of Michigan.

Finally, we thank John Dilley, Mikhail Mikhailov, Jake Brutlag, Flavia Peligrinelli Ribeiro, and the anonymous reviewers for carefully reading drafts and offering thoughtful, detailed comments.

10. REFERENCES

[1] V. Almeida, D. Menascé, R. Riedi, F. Peligrinelli, R. Fonseca, and W. Meira Jr. Analyzing Web robots and their impact on caching. In *Proc. 6th Web Caching Workshop*, June 2001.

[2] 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.

[3] H. Bahn, H. Lee, S. H. Noh, S. L. Min, and K. Koh. Replica-aware caching for Web proxies. *Computer Communications*, 25(3):183–188, Feb. 2002.

[4] P. Barford and M. Crovella. Generating representative Web workloads for network and server performance evaluation. In *Proc. ACM SIGMETRICS*, pages 151–160, July 1998.

[5] K. Bharat and A. Broder. Mirror, mirror on the Web: A study of host pairs with replicated content. In *Proc. 8th WWW Conf.*, May 1999.

[6] K. Bharat, A. Broder, J. Dean, and M. R. Henzinger. A comparison of techniques to find mirrored hosts on the WWW. In *Proc. Workshop on Organizing Web Space at 4th ACM Conference on Digital Libraries*, Aug. 1999.

[7] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *Proc. IEEE INFOCOM*, Mar. 1999.

[8] B. E. Brewington. *Observation of changing information sources*. PhD thesis, Dartmouth, June 2000. `http://actcomm.dartmouth.edu/papers/brewington:thesis.ps.gz`.

[9] B. E. Brewington and G. Cybenko. How dynamic is the web? In *Proc. 9th WWW Conf.*, May 2000.

[10] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the Web. In *Proc. 6th WWW Conf.*, Apr. 1997.

[11] J. Brutlag. Personal communication.

[12] R. Cáceres, B. Krishnamurthy, and J. Rexford. HTTP 1.0 logs considered harmful. Position Paper, W3C Web Characterization Group Workshop, Nov. 1998. `http://www.research.att.com/~jrex/papers/w3c.passant.ps`.

[13] CacheFlow Corporation. White paper: Creating a cache-friendly Web site, Apr. 2001. `http://www.cacheflow.com/technology/whitepapers/index.cfm`.

[14] 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.

[15] I. Cooper and J. Dilley. RFC 3143: Known HTTP proxy/caching problems, June 2001.

[16] 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.

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

[18] J. Dilley. Personal communication.

[19] J. Dilley and M. Arlitt. Improving proxy cache performance—analyzing three cache replacement policies. Technical Report HPL-199-142, HP Labs, Oct. 1999.

[20] F. Douglis, A. Feldmann, B. Krishnamurthy, and J. Mogul. Rate of change and other metrics: A live study of the World Wide Web. In *Proc. 1st USITS*, pages 147–158, Dec. 1997.

[21] A. Feldmann. Continuous online extraction of HTTP traces from packet traces. In *Proc. W3C Web Characterization Group Workshop*, 1999. `http://www.research.att.com/~anja/feldmann/papers.html`.

[22] A. Feldmann, R. Cáceres, F. Douglis, G. Glass, and M. Rabinovich. Performance of Web proxy caching in heterogeneous bandwidth environments. In *Proc. IEEE INFOCOM*, Mar. 1999.

[23] 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.

[24] D. Fisher. Personal communication.

[25] S. Gribble and E. Brewer. System design issues for Internet middleware services: Deductions from a large client trace. In *Proc. 1st USITS*, pages 207–218, Dec. 1997.

[26] A. Iyengar and J. Challenger. Improving Web server performance by caching dynamic data. In *Proc. 1st USITS*, pages 49–60, Dec. 1997.

[27] R. Jain. *The Art of Computer Systems Performance Analysis*. Wiley, 1991.

[28] T. Kelly. Thin-client Web access patterns: Measurements from a cache-busting proxy. *Computer Communications*, 25(4):357–366, Mar. 2002. `http://ai.eecs.umich.edu/~tpkelly/papers/`.

[29] M. Koletsou and G. M. Voelker. The Medusa proxy: A tool for exploring user-perceived Web performance. In *Proc. 6th Web Caching Workshop*, June 2001.

[30] B. Krishnamurthy and M. Arlitt. PRO-COW: Protocol compliance on the Web—a longitudinal study. In *Proc. 3rd USITS*, pages 109–122, Mar. 2001.

[31] Macromedia. Dreameaver, Nov. 2001. `http://www.macromedia.com/support/dreamweaver/`.

[32] P. Mattis, J. Plevyak, M. Haines, A. Beguelin, B. Totty, and D. Gourley. U.S. Patent #6,292,880: "Alias-free content-indexed object cache", Sept. 2001.

[33] M. Mikhailov and C. E. Wills. Change and relationship-driven content caching, distribution and assembly. Technical Report WPI-CS-TR-01-03, Worcester Polytechnic Institute, Mar. 2001.

[34] J. C. Mogul. Errors in timestamp-based HTTP header values. Technical Report 99/3, Compaq Western Research Laboratory, Dec. 1999.

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

[36] J. C. Mogul. Squeezing more bits out of HTTP caches. *IEEE Network*, 14(3):6–14, May/June 2000.

[37] J. C. Mogul, F. Douglis, A. Feldmann, and B. Krishnamurthy. Potential benefits of delta encoding and data compression for HTTP (corrected version). Technical Report 97/4a, Digital Western Research Lab, Dec. 1997.

[38] J. C. Mogul, B. Krishnamurthy, F. Douglis, A. Feldmann, Y. Y. Goland, A. van Hoff, and D. M. Hellerstein. RFC 3229: Delta encoding in HTTP, Jan. 2002.

[39] FAQ of the Caching Mechanism in [Mozilla] 331 Release, Apr. 2000. `http://www.mozilla.org/docs/netlib/cachefaq.html`.

[40] A. Muthitacharoen, B. Chen, and D. Mazieres. A low-bandwidth network file system. In *Proc. 18th SOSP*, pages 174–187, Oct. 2001.

[41] National Institute of Standards and Technology. Secure hash standard. FIPS Pub. 180-1, U.S. Dept. of Commerce, Apr. 1995. `http://csrc.nist.gov/publications/fips/fips180-1/fip180-1.txt`.

[42] H. Nordstrom. Squid cache revalidation and metadata updates. Posting to `squid-dev` mailing list, Oct. 2001. `http://www.squid-cache.org/mail-archive/squid-dev/200110/0054.html`.

[43] V. N. Padmanabhan and L. Qiu. The content and access dynamics of a busy Web server: Findings and implications. In *Proc. ACM SIGCOMM*, pages 111–123, Aug. 2000.

[44] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C*. Cambridge University Press, second edition, 1992.

[45] A. Rousskov and D. Wessels. The third cache-off: The official report. Technical report, The Measurement Factory, Inc., Oct. 2000. `http://www.measurement-factory.com/results/public/cacheoff/N03/report.by-meas.html`.

[46] J. Santos and D. Wetherall. Increasing effective link bandwidth by suppressing replicated data. In *Proc. USENIX Annual Technical Conf.*, June 1998.

[47] N. Shivakumar and H. Garcia-Molina. Finding near-replicas of documents on the Web. In *Proc. Workshop on Web Databases*, Mar. 1998. `http://www-db.stanford.edu/~shiva/Pubs/web.ps`.

[48] B. Smith, A. Acharya, T. Yang, and H. Zhu. Exploiting result equivalence in caching dynamic content. In *Proc. 2nd USITS*, pages 209–220, Oct. 1999.

[49] SPECweb. `http://www.spec.org/osg/web99/`.

[50] N. T. Spring and D. Wetherall. A protocol-independent technique for eliminating redundant network traffic. In *Proc. ACM SIGCOMM*, pages 87–95, Aug. 2000.

[51] D. Surovell. Personal communication.

[52] A. van Hoff, J. Giannandrea, M. Hapner, S. Carter, and M. Medin. The HTTP distribution and replication protocol. Technical Report NOTE-DRP, World Wide Web Consortium, Aug. 1997. `http://www.w3.org/TR/NOTE-drp-19970825.html`.

[53] Web Polygraph. `http://www.web-polygraph.org/`.

[54] C. E. Wills and M. Mikhailov. Examining the cacheability of user-requested Web resources. In *Proc. 4th Web Caching Workshop*, Apr. 1999.

[55] C. E. Wills and M. Mikhailov. Towards a better understanding of Web resources and server responses for improved caching. In *Proc. 8th WWW Conf.*, May 1999.

[56] C. E. Wills and M. Mikhailov. Studying the impact of more complete server information on Web caching. In *Proc. 5th Web Caching Workshop*, May 2000.

[57] 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 *Proc. 2nd USITS*, Oct. 1999.

[58] 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. Originally in SOSP '99.

[59] X. Zhang. Cachability of Web objects. Technical Report 2000-019, Boston U. CS Dept, Aug. 2000.