World Wide Web
DOI 10.1007/s11280-006-0224-x

Characterization of a Large Web Site Population with Implications for Content Delivery

Leeann Bent · Michael Rabinovich · Geoffrey M. Voelker · Zhen Xiao

Received: 13 August 2005 / Revised: 19 April 2006 / Accepted: 11 May 2006

Abstract This paper presents a systematic study of the properties of a large number of Web sites hosted by a major ISP. To our knowledge, ours is the first comprehensive study of a large server farm that contains thousands of commercial Web sites. We also perform a simulation analysis to estimate potential performance benefits of content delivery networks (CDNs) for these Web sites, and validate our analysis for several sites by replaying our trace through a real cache. We make several interesting observations about the current usage of Web technologies and Web site performance characteristics. First, compared with previous client workload studies, the Web server farm workload contains a much higher degree of uncacheable responses and responses that require mandatory cache validations. A significant reason for this is that cookie use is prevalent among our population, especially among more popular sites. We found an indication of widespread indiscriminate usage of cookies, which unnecessarily impedes the use of many content delivery optimizations. We also found that most Web sites do not utilize the cache-control features of the HTTP 1.1 protocol, resulting in suboptimal performance. Moreover, the implicit expiration time in client caches for responses is strongly constrained by the maximum values allowed in the Squid proxy. Thus, supplying explicit expiration information would significantly improve Web sites'

Bent, Rabinovich, and Xiao performed this work while at AT&T Labs-Research.

L. Bent (✉) · G. M. Voelker
University of California, La Jolla, 9500 Gillman Dr. MS-0114, San Diego, CA 92093-0114, USA
e-mail: lbent@cs.ucsd.edu

G. M. Voelker
e-mail: voelker@cs.ucsd.edu

M. Rabinovich
Case Western Reserve University, 10900 Euclid Avenue, Cleveland, OH 44106-7071, USA
e-mail: misha@eecs.case.edu

Z. Xiao
IBM T.J. Watson Research Center, 19 Skyline Drive, Hawthorne, New York, NY 10532, USA
e-mail: xiaozhen@us.ibm.com

cacheability. Finally, our simulation results indicate that while most Web sites benefit from the use of a CDN, the amount of the benefit varies widely among the sites, which underscores the need for workload analysis tools.

Categories and Subject Descriptors C.2.5 [Computer Communication Networks]: [Local and Wide Area Networks]-[Internet] · C.4 [Performance of Systems]: Performance Attributes · I.6 [Simulation and Modeling]: Applications

General Terms measurement · performance

Keywords web caching · content distribution · HTTP · workload characterization · cookie

1 Introduction

With the enormous growth of Web traffic on the Internet, various technologies have been proposed to optimize the amount of bandwidth consumption due to Web accesses, including caching, prefetching, and content delivery networks (CDNs). Understanding the characteristics of Web workloads is essential for evaluating the benefits of these various content delivery technologies. However, although there has been considerable effort characterizing Web client workloads at many scales and locales (e.g., [12–14, 18, 26, 32]), there has been little previous work characterizing Web site workloads across a large numbers of Web sites. Previous work in this area has either considered a small set (under a dozen) of hand-picked Web sites [3], or a single high-volume Web site [2, 16, 25, 27]. While these studies have been valuable for certain purposes, a comprehensive study of a broader, more representative population of Web sites can provide important insights for research in content delivery as well as Web site design.

In this paper, we present a systematic performance study of a large number of Web servers: We examine the traffic to three thousand commercial Web sites hosted on a large server farm by a major Internet service provider. Unlike previous proxy-based studies that focus on traffic analysis from a client/cache point of view, we present a server-eye view of the properties of these Web sites. We also perform a CDN simulation analysis to estimate potential performance benefits a Web site might see from subscribing to CDN services.

We make several interesting observations about the current usage of Web technologies and Web site performance characteristics. Compared with previous client workload studies, the Web server farm workload contains a much higher degree (66% of responses) of uncacheable responses and responses that require mandatory cache validations. A significant reason for this is that cookie use is prevalent in our workload (47% of requests), especially among more popular sites. Further, we found an indication of widespread indiscriminate usage of cookies, which impedes many content delivery optimizations unnecessarily. We also found that most Web sites do not utilize the cache-control features of the HTTP 1.1 protocol, resulting in suboptimal performance. Moreover, the effective expiration time of most of their responses in client caches (the "time-to-live" or TTL) is constrained by the maximum values allowed in the popular Squid proxy. This indicates that Web sites would improve their cacheability significantly by supplying

explicit expiration information for their content. Finally, our simulation results indicate that while the Web sites benefit from the use of a CDN, the amount of the benefit varies widely among the sites. Thus, individual sites must make a decision about using CDN services by analyzing their particular workloads, and there is an urgent need for tools that would simplify this task.

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 presents our trace methodology. Section 4 describes general properties of our server population. Section 5 presents a per Web site analysis, and Section 6 studies the benefits of using a CDN for the Web servers. Finally, Section 8 concludes.

2 Related work

A summary of Web characterization studies can be found in [28]. Many studies examine the performance of a small set (on the order of a dozen) of Web sites in detail, including [3, 9, 24]. Other studies present a careful analysis of a single very high-volume site [2, 16, 27]. Unlike these studies, ours is a systematic study of a large number of commercial Web sites, focusing on the busiest 3,000 sites of a Web site population totaling over 34,000 sites. Furthermore, rather than manually choosing individual sites to group together into a study, we use the natural grouping afforded by the Web farm to study a combination of sites that customers are willing to pay to have hosted and users want to visit. As a result, we believe our study provides a more representative view of the properties of "run-of-the-mill" commercial sites. Another difference is that some previous work (e.g., [27]) only considers accesses to root pages, while our analysis includes accesses to all objects.

Krishnamurthy and Arlitt [21] and Krishnamurthy and Wills [20] examine accesses to many Web sites. However, [21] focuses primarily on protocol compliance, while [20] focuses on persistent and parallel connection usage. In addition, both papers study a far smaller number of Web sites than our study and only consider root pages.

Many studies analyze accesses to a large number of Web sites from a client-centric point of view (e.g., [5, 33]). While client-centric studies provide insights into the behavior of a given set of clients, only a study from a server perspective, such as a large server farm, can fully capture the set of events that happen at the server.

Various studies report some of the characteristics that we study, albeit from a client perspective. Feldmann et al. [13] reported the frequency of cookie occurrence and cacheability of Web content. Wills and Mihkailov [31] focused on content cacheability. Using a set of over a thousand URLs selected from a proxy log, they observed that many requests to images carry cookies that are functionally unnecessary: the cookies do not affect responses. This observation has significant implications to the Web sites in our study.

The frequency of modifications to Web pages was considered by Douglis et al. [11] and Brewington and Cybenko [8]. We do not focus on this characteristic, but it is indirectly relevant to our study because it affects the prevalence of "Not-modified" responses.

Several studies consider CDN benefits. Krishnamurthy et al. [22] concentrate on a comparative performance study of different CDNs. Unlike our work, Krishnamurthy et al. are interested in the download time of pre-selected pages through various CDNs, whereas we consider the CDN effects on a Web site. Jung et al. [17] investigate the

ability of a CDN to protect a Web site from a flash crowd. While they study known flash events that occurred on two Web sites, we consider a large set of Web sites during a random 20 h time period.

Ranuak et al. [29] studied the benefits of proxy caches on the tail of the load distribution. They found that a proxy cache benefits the peak bandwidth intervals much less than average bandwidth intervals and conjecture this is due to poor locality at the peak. We find similar results for CDN usage considering CDN hit rate.

3 Methodology

Our trace consists of 21 h of the first TCP data packet of HTTP requests and responses on one up-link into and out of a Web server farm from a large commercial hosting service of a major ISP. We captured all packets that begin with "GET" to the Web sites and all packets that begin with "HTTP" from the web sites. The trace was gathered using the Gigascope appliance [10] from 11:00 P.M. on July 14, 2003, to 7:49 P.M. on July 15, 2003. It contains 41,943,804 requests and 38,828,393 responses comprising 47 GB of total data before processing (the numbers of requests and responses do not match primarily because we only monitor one up-link; due to traffic management, some responses are sent back on different links than the one carrying the request). Because our trace interleaves separate request and response records, we have to match each request with the appropriate response. After matching requests and responses, we post-processed the trace for analysis.

3.1 Matching requests and responses

Our trace contains interleaved records for requests and responses. To examine the data as HTTP request/response transactions, we must match requests with responses. We perform this matching based on the five-tuple of source IP address, destination IP address, source port, destination port and time. We then sort request/response pairs by request time. After matching the requests with responses, the trace contained 26,136,345 request/response pairs.

To identify matching errors, we compared the type of the object request with the type of the object response. We used the URL suffix as a heuristic for the request object type, and the response type in the HTTP header for response object type. We have excluded control responses from this comparison, as many of these have no type (e.g., "304 Not Modified" is often given without object type) or a fixed type (e.g., "404 Not Found" returns HTML regardless of request type).

To validate our matching procedure, we tested one and a half hours from the trace. We found a response type for more than half of the responses (57%), and found a 1.4% mismatch rate after the post-processing step described below. Since this error rate is reasonably small, we considered our matching procedure accurate enough to perform our analyzes.

3.2 Post-processing

To gather commercial Web site statistics from the trace, we post-processed the request/response pairs. We excluded all but the top 3,000 sites. We excluded unpopular sites because they had a request rate that was too low—fewer than 485

requests per site over the course of the trace—to draw meaningful conclusions. These top 3,000 sites service 90% of request/response pairs in the trace.

We also removed requests/response pairs that could not be mapped to a single Web site. Since some IP addresses in our study serve multiple Web sites, we cannot use IP address as an indication of Web site. Instead, we use the `Host` field given in the HTTP header when available. Below we discuss how we use this field in more detail.

Note that we define *hostname* to be the actual host listed in the `Host` HTTP header field, while we define *Web site* to be a set of hostnames sharing the last two components of their domain names. For example, `www.firm-x.com` and `images.firm-x.com` are hostnames associated with the Web site `firm-x.com`. Because we are using the `Host` field to identify Web sites, we removed all requests that did not identify a hostname in the `Host` field. There were 90,301 (0.3%) request/response pairs in the trace with no `Host` field. Further, pipelined request/response pairs were removed. We identified these by looking for request packets with multiple `GET` requests in them. While pipelined request/response pairs may contain a valid match (the first `GET` request may be a correct match with the response), the remaining `GET` requests will have no match. Since there were very few of these we have removed them, along with requests that contain multiple `Host` HTTP headers. The total number of request/response pairs that were removed due to pipelining or multiple host names is 1,510 (less than 0.1% of request/response pairs).

After this, we identified *Web sites* by first cleaning up the `Host` HTTP field (discarding those that could not be cleaned) and then by associating *hostnames* with *Web sites*. We found that some request/response pairs have given the `Host` HTTP header as an IP string. Where appropriate (i.e., there was a unique mapping of one hostname to the IP address from elsewhere in the trace), we mapped the IP address to that hostname. `Host` HTTP headers where there was no mapping were discarded. There were 153,355 (0.6% of request/response pairs) of these. We also found that there were request/response pairs with partial hostnames (because the packet had been cut off). If a hostname is a proper prefix of another hostname it is deemed a partial hostname. Requests with partial hostnames were also removed. There were 6,468 such request/response pairs (less than 0.1% of the total). Finally, hostnames with unprintable characters (there were only 27 of these) were removed.

Once the `Host` field had been cleaned, it was used to group the hostnames into Web sites using the last two components of the hostname as described earlier.

Overall, out of 26,136,345 matched request/response pairs, 23% are removed from the trace due to contractual restrictions. Further cleaning, including restricting the trace to 3,000 most popular sites, removes 11% of remaining requests, leaving 17,818,437 request/response pairs in the cleaned trace.

4 Trace properties

We start our analysis by summarizing overall characteristics of the workload in our trace. After post-processing, the trace contained 17,818,437 matched request/response pairs to 3,000 Web sites. Figure 1 shows these requests to the Web sites over time at the granularity of 10 min. This time series exhibits the diurnal pattern exhibited by typical Web workloads. The average request rate per Web site was 5,939 requests over the lifetime of the trace, or 282 requests per hour. The Web sites

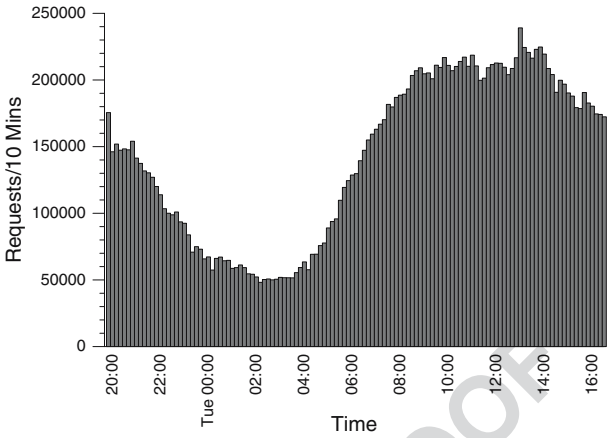


Figure 1 Requests to the 3,000 most popular Web sites over time for the duration of our trace.

in the trace span a wide range of activity, with the most popular site receiving 2.1 million requests and the least popular receiving only 485. In Section 6, we explore the topic of origin server load by studying the effects of using a CDN with this Web server farm.

Table 1 shows various overall properties of our trace. In terms of protocol version, we found that 14% of requests used HTTP/1.0 and 86% of requests used HTTP/1.1. For those objects with a non-zero `Content − Length` header, their average response size was 9 KB.

Table 2 shows requested object types. Most requests (77%) were to images (URLs with file extensions of `.gif`/`.jpg`/`.jpeg`), 5.2% were to HTML (URLs with file extensions of `.html`/`.htm`) or base pages (i.e., /), 8.6% are CGI, and very few (0.2%) are documents (URLs with file extensions of `.ps`/`.doc`/`.ppt`/`.pdf`). We identified CGI objects as those objects whose URL contains the `/cgi` substring (used as an indication of the `/cgi − bin/` path or other cgi directory), contains a question mark (`?`), or ends in `.asp`, `.aspx`, or `.cgi`. We call will refer to this later in the paper as the *URL substring heuristic*.

Compared to client workloads, this server farm workload contains a much higher concentration of requests to images (77% in our workload compared to 54% in [32]). We speculate that the nature of the commercial Web sites results in a larger percentage of image content compared with all Web sites accessed by a large client population. The preponderance of image requests has substantial caching implications for our workload, and we discuss this issue in detail in Section 5.1 below. Also,

Table 1 Overall trace properties.

Property	Value
Request/response pairs	17,818,437
Trace size	33 GB
Number of clients	376,678
HTTP/1.0 downloads	14%
HTTP/1.1 downloads	86%
Average object size	9 KB

Table 2 Request object type popularities.

Object type	Prevalence (%)
Images	77
HTML	5.2
CGI	8.6
.doc,.pdf,.ppt,.ps	0.2
Audio	0.1
Other	9.1

the average response size in our trace is significantly lower than in previous studies (9 KB in our trace compared with, e.g., 15 KB in [5]). This effect partly can be explained by a large number of small embedded images.

Table 3 shows the relative prevalence of response codes in the trace. The majority (64%) are `200 OK`, and most of the remainder (29%) are `304 Not Modified`. We found only a negligible number of `5XX Server Error` responses. Note that `304` responses are caused by validation requests that ask for the content only if it has changed from the version cached by the requester. The large percentage of `304` responses indicates that many of these validation requests were in a sense unnecessary because the requested object at the server has not been modified, and the cached object could have been used. We explore this issue in more detail in Section 5.2 below.

Finally, an interesting question is how often requests to our Web sites come through cache servers. According to HTTP/1.1., cache servers must include a `Via` header as they forward client requests upstream. While it is unclear how often this provision is adhered to, the `Via` header providers a lower bound on the number of requests coming through cache servers. About 8.5% of requests in our trace include the `Via` field. Additionally, we found that 2,511 of 3,000 (83%) Web sites see at least one request from a downstream cache. Thus, while proxy caching or forwarding is not prevalent among our client population, most sites do indeed see some form of downstream caching. Thus, while proxy caching or forwarding is not prevalent among our client population, most sites do indeed see some form of downstream caching. Thus, Web sites must use cache-controlling features of the HTTP protocol to handle downstream caches properly.

5 Web site characteristics

One of the main goals of our study was to characterize a large population of Web sites, particularly with respect to the properties relevant to content delivery. We focus on metrics that influence performance optimizations and may indicate

Table 3 Response code popularities.

Response type	Prevalence (%)
Ok (200)	64
Not modified (304)	29
Found (302)	2.6
Client error (4XX)	3.9
Other	0.4

potential performance problems within a web site. We do this by analyzing behavior on a per web site basis. Average behavior gives us intuition about the behavior of a commercial web site. Outliers from any average metric are good candidates for performance problems. Per web site statistics will show where we get the biggest win for potential performance improvements, on a per server basis and from a "network" point of view. Finally, the per web site analysis highlights unexpected behavior of web servers and unintended consequences of web server policies.

5.1 Cookie usage

We start by studying the usage of cookies by the Web sites in our trace. A cookie is a piece of data that the server includes with its response to the client and which the client sends back to the server in subsequent requests. Cookies are widely used in today's Web to personalize content, to track user browsing within the site, to prevent unauthorized access, etc. To store a cookie on a client, the server uses a `Set-Cookie` HTTP response header. Subsequent requests will carry the cookie in a `Cookie` HTTP request header. Besides the cookie value, the `Set-Cookie` header includes `domain` and `path` attributes, which determine the URLs that are relevant to this cookie. Only requests for URLs with host names belonging to the specified domain and with URL path prefixes matching the path attribute will include the corresponding cookie header.

Cookie usage has important implications for content delivery. Cookies are often used for page personalization or for e-commerce, and thus many caches do not satisfy "cookied" requests (i.e., requests that carry a `Cookie` header) from caches, be it forward proxies or CDN edge servers. Instead, these caches send `If-Modified-Since` requests with the same cookie to origin servers before sending their cached responses to clients.

Analyzing precise impact of cookies on content cacheability is difficult because, in the absence of HTTP 1.1-specified behavior with respect to cookies, different cache products treat cookies differently. While old versions of Squid simply tunneled cookied requests through to the origin server (thus negating any cache benefits for cookied requests), newer versions converted these requests to conditional `If-Modified-Since` requests [31], and version 2.5 that we tested seems to ignore cookies altogether. On the other hand, a leading commercial cache (Network Appliances' NetCache) by default always validates objects requested with cookies, and does not cache responses with `Set-Cookie` header at all (we tested version 5.6). Ignoring cookies improves cache benefits but often leads to incorrect behavior, with personalized pages returned to unintended recipients.[1]

Although different caches treat cookies differently, it is obvious that cookies complicate content delivery. In this study, we assume that a cookied request from a browser must always reach the origin server either as a full request or as an `If-Modified-Since` request. Because of this "cache busting" effect, cookies must be used judiciously to fully exploit Web caching infrastructure.

[1] Strictly speaking, caches are not responsible to make any allowances for cookies according to HTTP 1.1. However, Web sites in practice often expect them to do so and neglect to include explicit cache controlling headers that could enforce correct behavior.

5.1.1 Frequency of cookie use

We find that cookie use in the requests in our trace is widespread: 47% of requests contain a cookie. Furthermore, cookie use is skewed toward popular web sites. While 47% of requests contain cookies overall, only 35% of servers receive requests with cookies in them. This implies that those 35% of servers receive a disproportionally high share of requests. To illustrate this trend in more detail, Figure 2 splits all sites into popularity bins, where the left-most bar corresponds to the most popular 1% of web sites, the next bar corresponds to the next most popular 1% of web sites, and so forth. The bar heights correspond to the percentage of cookied requests directed to the corresponding groups of servers. This graph clearly shows that cookie use is skewed toward the more popular sites, especially the most popular 1% of Web sites. We also performed a correlation analysis between the site popularity (measured in the number of requests to the site in the trace) and the prevalence of cookies. When considering the 3,000 most popular sites, we found a low positive correlation of 0.09. Analyzing just the 279 sites with at least 5,000 requests, we found the significant positive correlation of 0.18.

In summary, 47% of our workload is not fully cacheable simply because they have a cookie attached. Our next question is if this high usage is inherently necessary. Obviously, discerning the intention of Web site designers is imprecise science, and so answering this question authoritatively is difficult. However, we can evaluate the extent to which cookie use could be changed to improve the delivery of a site's content. We consider two closely related aspects as an indication of unnecessary usage.

First, we consider cookied requests to images as an indication of unnecessary use (although see Section 5.1.4 for exceptions to this rule). Wills and Mikhailov [31] observed that almost 90% of cookied requests for images return responses that do not depend on the cookies and concluded that cookies in most of these requests are not inherently necessary. Indeed, if not for image personalization, cookies in image requests could also be used to track accesses by a given client or for access control. But this tracking is redundant for embedded images since their accesses can be in-

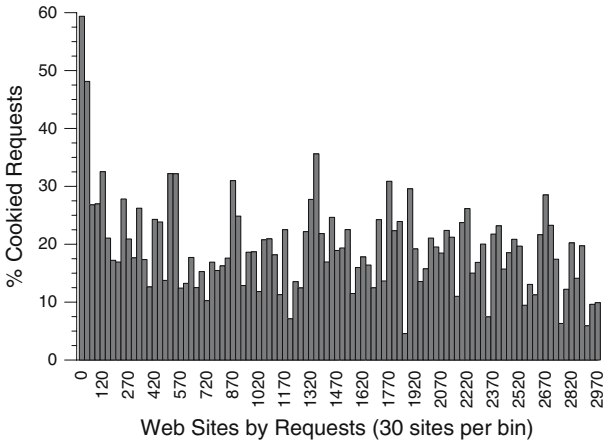


Figure 2 Prevalence of cookie use by server popularity.

ferred from the accesses of their containing page. Only when the image is hyperlinked to a page and must be clicked on to be viewed can cookies be justified. Intuitively, images are more often embedded than hyperlinked (although sites serving primarily image collections such as adult sites are a notable exception to this rule).

Second, we consider responses that set cookies for all requests to any URLs on the site as an indication of injudicious cookie use. We conjecture that these non-specific `Set − Cookie` headers are largely responsible for the prevalence of cookied image requests mentioned earlier. In these sites, only the initial requests from a given client do not carry cookies. It is hard to imagine that every request to a site, including all applets, Javascript modules, images, etc., requires a cookie.

5.1.2 Cookied requests for images

Figure 3 shows the prevalence of cookied requests for images. The graph includes the approximately 1,000 sites that use cookies. The *x*-axis shows the Web sites ranked in the order of their percentage of cookied requests that are image requests out of the total number of cookied requests, and the *y*-axis shows the percentage of cookied requests to that site that access images. We see a wide range of prevalence of cookied image requests. However, clearly a large number of sites set cookies so that they apply to images. This indicates a high occurrence of unnecessary cookie use: 73% of all cookied requests are for images and, overall, 34% of all requests are for cookied images. In other words, more judicious use of cookies would likely cut their occurrence by more than half, substantially improving the effectiveness of CDN and client-side caching.

In fact, we found that cookied image requests constitute a large portion of *all* requests received by those sites that use cookies. Figure 4 shows the percentage of all requests received by sites that were requests to images with cookies. The *x*-axis plots site rank in decreasing order of the above percentage for those sites that use cookies, and the *y*-axis shows the percentage value. We see that, for a large number of sites, cookied image requests represent the majority of the overall requests these sites receive. Given that images are responsible for a majority of requests on the Web in general (see, e.g., [4, 18, 33]), we can explain this result if we assume, again,

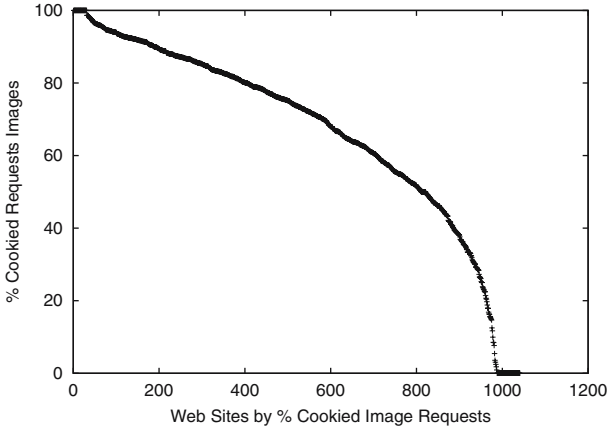


Figure 3 Prevalence of cookied requests for images.

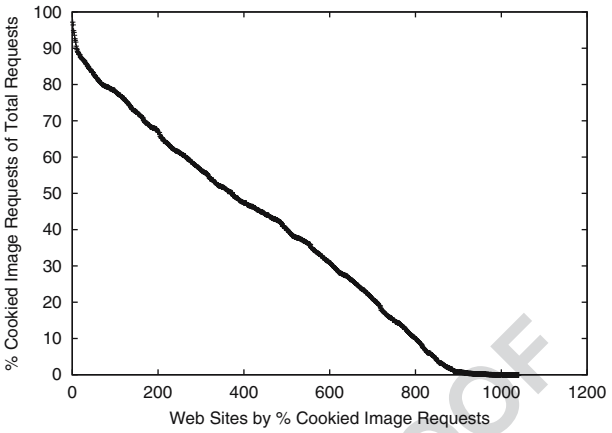


Figure 4 Percentage of cookied image requests over all requests to sites that use cookies.

that the main reason for the cookied images is that sites simply set cookies indiscriminately for all their URLs. But it also shows that, by doing so, these sites deny themselves much of the performance benefits from CDNs and Web caching.

The left-most point of Figure 4 is particularly interesting as it shows that practically all requests to this site were cookied images. A closer look revealed that this is a credit services site and most of the objects on the site are cookied images. Many of these images, in fact, appear to be navigational objects (e.g., menu bars) or spacer images in a menu or logo. Again, it seems unlikely that all of these images require cookies.

We also consider how indiscriminate cookie use depends on site popularity. While we showed that popular sites use cookies more often, we do not find that they use cookies more appropriately. Figure 5 shows how cookied image requests correspond to object popularity. On the *x*-axis, we show the number of requests received by each Web site and on the *y*-axis we show the percentage of cookied requests that are requests to images. As the outlying points of the graph show, even very popular sites (e.g., the sites with between 500 K and 2.1 M requests) return many cookied images.

As mentioned earlier, some clients mitigate the content delivery limitations of using cookies by issuing `If-Modified-Since` (IMS) requests for objects that they already have in their caches. Figure 6 shows that there are a large number of these

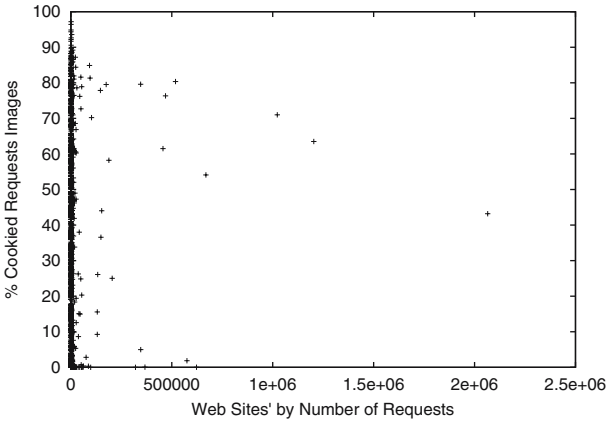


Figure 5 Prevalence of cookied requests for images.

requests. The *y*-axis shows the percentage of cookied image requests received that are IMS requests. The *x*-axis shows the Web sites that receive IMS requests for cookied images in decreasing order of percentage. Over 85% (884) of sites that use cookies receive at least one `If − Modified − Since` cookied image request. 14% of total requests are cookied IMS image requests. Recall that 34% of all requests to a site are cookied image requests. Then 20% of all requests were cookied image requests resulting in a full download. In addition, although the 14% requests with the IMS header do not consume much bandwidth, they still impact performance because they increase client latency and origin server load.

5.1.3 `Path` *and* `Domain` *attributes*

The source of the injudicious cookie use can be traced to how sites specify `Set − Cookie` headers. Almost all `Set − Cookie` headers in the trace contain attributes that actually widen the applicability of a corresponding cookie from the path which set the cookie to all objects on the site.

For example, nearly all `Set − Cookie` headers contain the `path` attribute specifying the root path "/". With this attribute, any request to the host that set this cookie must carry the cookie. Notice that without a path attribute, the cookie would have only applied to requests for URLs that share the path with the URL that had set the cookie.

Only 17 Web sites (out of around 750 Web sites we found using the `Set − Cookie` header) specify restrictive path attributes in `Set − Cookie` headers. Even these sites often provide these restrictive paths only in some responses and use root paths in other cases. Figure 7 shows, for each of these 17 sites, the percentage of `Set − Cookie` headers with restrictive paths. We see that only three sites consistently specify restrictive paths for their cookies, indicating that only they use cookies judiciously. A fourth site almost always specifies restrictive paths (99.8%), but even one non-restrictive path means that there is one cookie that applies to the whole site for at least one client.

Focusing on the `domain` attribute, only 16 sites specify the domain attribute in `Set − Cookie` headers. The vast majority rely on the default instead, meaning that

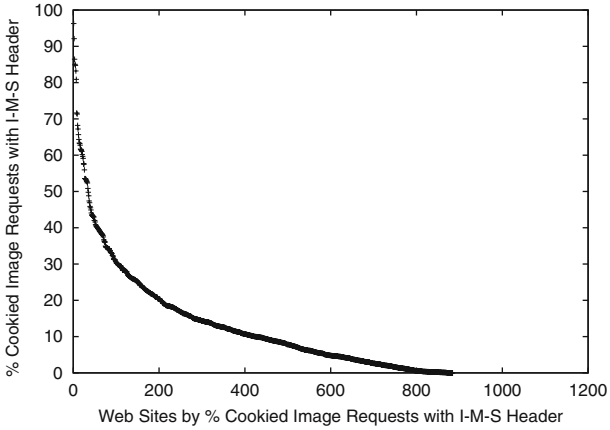


Figure 6 Percentage of `If − Modified − Since` cookied requests for images over all requests to sites that use cookies.

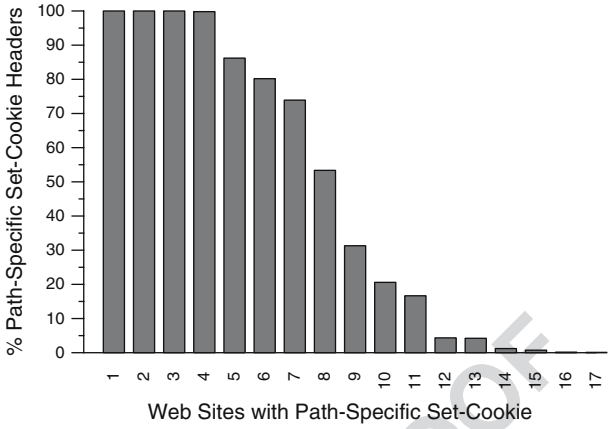


Figure 7 Prevalence of `Set – Cookie` header with restrictive path specifier.

the cookie will apply only to the host that set the cookie. For instance, if the cookie was set by `www.firm – x.com`, it will not apply to URLs with host name `images.firm – x.com`. One could in principle use this default rule to restrict cookie use by grouping all resources requiring cookies into distinct domain names from other resources. However, most Web sites (2,752 sites, or 92%) in this trace use only one host name for the entire site, and providing or omitting the domain attribute does not affect cookie applicability.

Of these 16 sites, only 4 sites specify a domain that is a specific, 3-level domain. In fact, 12 of the 16 sites that do provide a domain attribute in their `Set – Cookie` headers appear to use it to widen the applicability of the cookie. Similar to the path attribute, these sites specify the most general domain they can (using only the two top-level domain names, such as `firm – x.com`), thus requiring requests to all subdomains to carry the cookie. Again, this is an indication of indiscriminate cookie use.

5.1.4 Discussion of cookie usage

Our results show that cookies are used indiscriminately in the sites that we studied, and that this usage significantly limits content delivery performance. How can sites fix this problem?

A direct approach for restricting the use of cookies to the content that needs it is to use the site name space to separate cookied content. For example, since most images typically do not require cookies, placing them in a separate path (e.g., "/images") or on a different domain (e.g., "images.domain.com") easily separates them from content that may require cookies (e.g., HTML container pages that require secure access or tracking). With this approach, the site can easily specify the path and domain attributes in the `Set – Cookie` header to only use cookies for such content.

However, separating content using the name space may be inconvenient to the developer of the content. In terms of organizing and maintaining content, for example, it may be most convenient to organize embedded images with the HTML pages that contain them. In this case, manually using `Set – Cookie` to pinpoint cookie use would likely be burdensome. Since many commercial sites create their content using Web site authoring tools, we see this as an opportunity for such tools to assist content developers in managing cookie use for their content.

Of course, some sites *do* require cookies for much of their content, including image content. For example, one of the sites in our trace is an adult content site whose requests were almost entirely cookied images. For our purposes, the important characteristics of such sites are that:

- They have a high occurrence of images that are hyperlinked rather than embedded in HTML pages. In other words, the browser does not download these images automatically—a user must click on a link to download an image.
- The site needs to keep track of per-client usage of the hyperlinked images to discern preferences of a given user.

Besides adult sites, museum sites and other sites with many hyperlinked images (e.g., NASA) may belong to this category. Still, we argue that even for these sites setting indiscriminate cookies for the entire site is not necessary. For example, in addition to the hyperlinked images, the NASA site contains many embedded images that do not need to be tracked because their usage can always be inferred from the usage of their containing HTML page. Again, tracking only the hyperlinked images can be achieved organizationally by placing them in a separate directory path and setting cookies for this path only.

5.2 Cacheability properties

Cacheability of content plays a pivotal role in the effectiveness of the various traditional content delivery techniques that focus on storing static objects in the network for future use. These techniques include both client-side proxy caching and server-side CDNs and Web accelerators.

5.2.1 Caching headers

HTTP/1.1 provides to the Web site developer expressive headers that allow fine-grained control of caching behavior. These headers are responsible to a large extent for the complexity of the protocol and of the cache and CDN servers. However, we found that neither client browsers nor Web sites make much use of these headers. Table 4 shows the percentage of requests that use any of the various cache-controlling headers available to them (note that the sum of all header and header directives do not add up to the 'Any' header line due to rounding errors). Table 5 shows the same for Web sites and responses. We find that, among requests, the `Cache-Control` values of `max-age` and `max-stale` are most prevalent, both accounting for 2.5% of requests. Of response cache controlling headers, we find that `no-cache`, specified either as a `Pragma` value or as a `Cache-Control` value is the most widely used, accounting for 4.7% of responses. However, all of the cache-

Table 4 Usage of cache-controlling headers in requests.

Header or header directive	Prevalence (overall; %)
Max-age	2.5
Max-stale	2.5
No-cache	2.2
Other	0.1
Any	6.8

Table 5 Usage of cache-controlling headers in responses.

Header or header directive	Sites using the header	Prevalence (among using sites; %)	Prevalence (overall; %)
No-cache	175	9.1	4.7
Private	726	6.2	2.1
No-store	30	4.4	2.0
Expires	215	10	3.3
Revalidate	32	9.2	3.9
Age	2	33	0.2
Max-age	18	17	2.0
No-transform	92	9.2	0.1
Other	27	3.0	1.7
Any	838	7.7	9.3

controlling headers are used very little and only 6.8% of all requests and only 9.3% of all responses use *any* sort of cache control headers. In addition, the most prevalent response caching header is used to *deny* cacheability.

Table 6 shows the use of additional response headers that may affect response caching in the trace. The `ETag` header is used to validate cached objects. We see widespread usage of the `ETag` header, with almost all sites using the header and 82% of responses overall. We see little use of two other cacheability-affecting response headers: `Transfer − Encoding : chunked` and `Content − Range`. We found that both response types were supplied by a large number of Web sites (1,306 and 1,949, respectively), but were almost unused overall, with chunked encoding accounting for 4.1% of responses and range responses accounting for 0.3% of responses. `Content − Range` is another cacheability-affecting response header, driven by client browsers requesting a part of an object. For example, Squid does not cache range responses. We found that range responses were supplied by a large number of Web sites, but were very rare, accounting for only 0.7% of responses among the sites that ever used them and 0.3% of overall responses.

More extensive use of cache-controlling headers could improve cacheability, decreasing load at the Web site. In the absence of these headers, caches tend to be very conservative in what they cache. Hence, these Web sites defeat to a large extent traditional content delivery technologies from which they might otherwise have benefited, including reduced bandwidth consumption and server load.

Sites may not use these headers either because content developers are not aware of their existence, are not aware of the potential benefits of using them, or find that

Table 6 Usage of cache-affecting headers in responses.

Header or header directive	Sites using the header	Prevalence (among using sites; %)	Prevalence (overall; %)
ETag	2989	89	82
Chunked	1306	7.1	4.1
Range response	1949	0.7	0.3

the complexity of using them is too burdensome. As with cookie usage, this situation presents another opportunity for Web site authoring tools. Such tools already manage the complexity of the content itself, and could also help manage the complexity of using the cache control directives to maximize content cacheability and improve content delivery performance.

5.2.2 *Use of* `Last-Modified` *Header*

A header that is not strictly for cache control but that has a profound effect on caches is `Last-Modified`, which specifies the time of the last update to the object. In the absence of an explicit expiry time, caches use the `Last-Modified` header to compute heuristically how long they can cache the response before validating it with the origin site. When there is no `Last-Modified` header, caches cannot use this heuristic. Generally, caches assume responses without the `Last-Modified` header were dynamically generated and conservatively do not cache them.

While almost all Web sites supply the `Last-Modified` header, the frequency of its use varies widely. Figure 8 shows the percentage of a site's responses without this header. Overall, a large fraction of responses (44%) lack this header. And, on average, 34% of responses from each site do not have the `Last-Modified` header. We also examined `Last-Modified` usage according to Web site popularity and found no correlation between Web site popularity and use of the `Last-Modified` header.

The large number of requests without `Last-Modified` may in itself indicate low content cacheability. However, a more detailed analysis shows that many responses with no `Last-Modified` header are "Not Modified" control messages that validate cache objects and do not affect content cacheability (see Figure 9). On average, 67% of responses without a `Last-Modified` header were "Not Modified" control messages (both on a per Web site basis and across all sites).

Because Web servers supply the `Last-Modified` header for static files automatically, we speculate that the lack of this header in a response (that is not a control message like "Not Modified") indicates a dynamically generated response.

Many studies use the heuristic for identifying CGI objects as in Section 4, which we refer to as *URL substring heuristic*. Figure 10 plots the percentage of responses

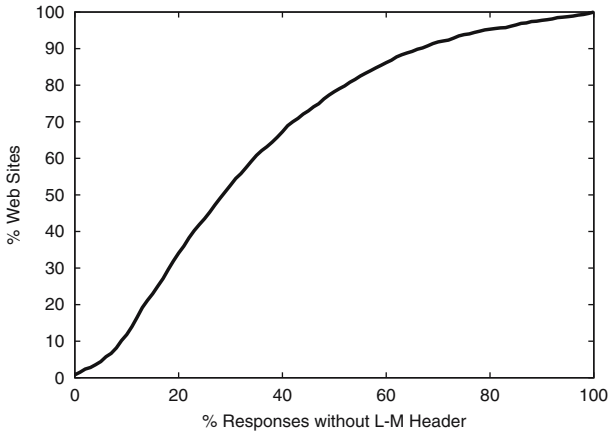


Figure 8 Prevalence of responses without the `Last-Modified` header.

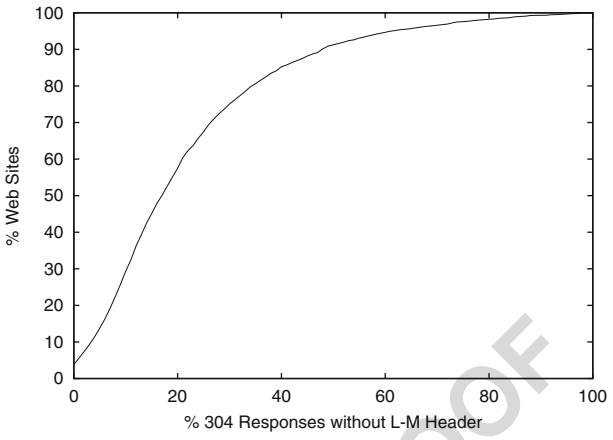


Figure 9 Prevalence of 304 response.

classified as dynamically generated according to the URL substring heuristic. Overall, only about a third of all sites (1,195) return such responses, and, among these sites, these responses represent on average 4% of all responses from a given site and 9% overall. Recall that 29% of responses were 304 responses with no `Last – Modified` header, leaving 38% of total responses without `Last – Modified` headers accounted for. However, this implies that 6% of responses overall still have no `Last – Modified` header. These responses are not 304 responses and are not classified as dynamic responses by the URL substring heuristic.

Thus the URL substring heuristic undercounts some dynamic responses. To explore how this under-counting affects individual sites, Figure 11 plots the distribution of 200 responses without the `Last – Modified` header that would not be identified as dynamic by the URL substring heuristic (which we refer to as "non-cgi" responses for short). For over 98% of Web sites, less than 10% of their overall responses are non-cgi 200 responses with no `Last – Modified` header. While this result seems to indicate low under-counting, it is in fact quite significant compared

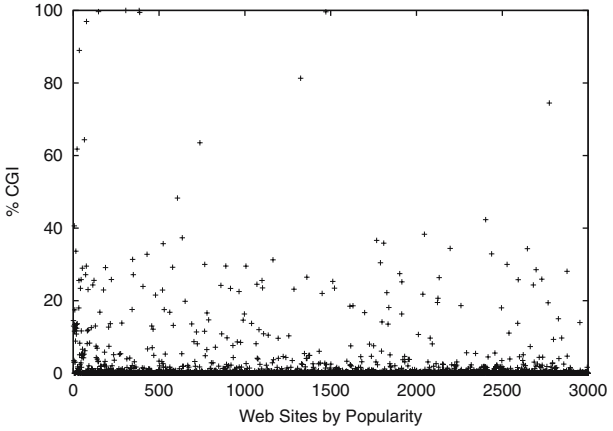


Figure 10 The percentage of requests to CGI objects for each Web site in decreasing order of site popularity (according to the URL substring heuristic).

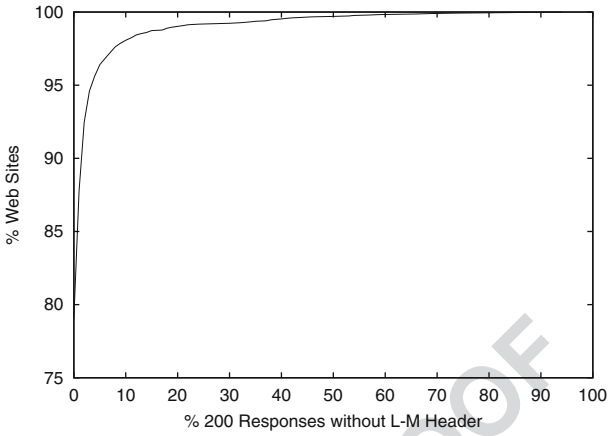


Figure 11 Prevalence of "200" responses other than CGI that have no `Last – Modified` header.

to the number of requests identified as dynamic by the URL substring heuristic. Moreover, a handful of sites (ten) received more than 50% of non-cgi, 200 responses without the `Last – Modified` header.

5.2.3 Cacheability

Finally, we consider the overall frequency of responses that are not fully cacheable (i.e., responses that cannot be served from a cache without contacting the origin server). In our study, request/response pairs are considered not fully cacheable if (1) either the request or response cache-controlling headers indicate so, (2) if the response contains no `Last – Modified` header, (3) the request contains a `Cookie` header or (4) the response contains a `Set – Cookie` header. Following previous studies (e.g., [14]), we will call them "uncacheable" although, as discussed earlier, they may be only cache validations. Figure 12 plots the CDF of overall percentage of uncacheable responses. Note that a response can be uncacheable because of either the request or the response in the request/response pair. We found that a large portion of Web content from the sites in our trace is uncacheable: overall, 66% of responses across all the sites are uncacheable. Somewhat surprisingly, this degree of uncacheability is substantially higher than that reported in previous client and proxy caching workload studies: Feldmann et al. [13] report 38–43% uncacheable responses from workloads of dialup modem clients at a large ISP and clients on a fast LAN, and Wolman et al. [32] report 40% uncacheable responses from a large university client workload. Per site, 45% of responses are uncacheable on average and 50% of Web sites have 36% or more responses uncacheable. (Similar to us, both studies considered CGI objects and cookied requests and responses uncacheable.) We attribute this higher degree of uncacheability to our focus on commercial Web sites and substantial use of cookies.

Another conclusion is that CDNs have an opportunity to increase their benefits by a closer cooperation with content providers. For example, the content provider can explicitly invalidate cached objects in CDN caches, reducing the need for "Not Modified" responses without sacrificing data freshness. Also, by understanding the nature of the content, CDNs can be more aggressive in deciding what content can be

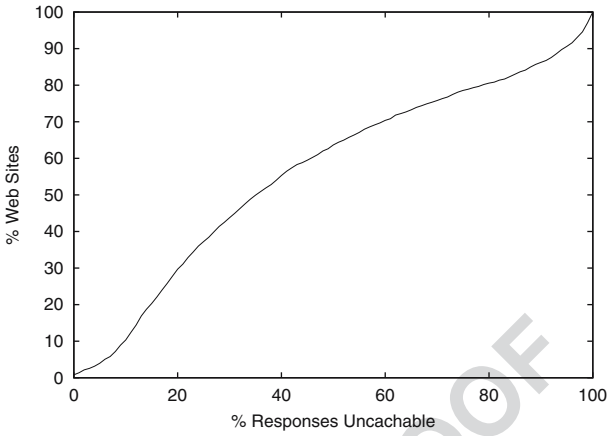


Figure 12 CDF of percentage of responses to a Web site that are uncacheable.

cached. Finally, emerging technologies aimed at accelerating dynamic content and Web applications should further increase CDN benefits.

5.3 Cache consistency-related properties

Figure 13 shows the cumulative distribution of the average TTL for the top 3,000 Web sites. For all cacheable objects, we compute the TTL value for a response message based on the age and `Last – Modified Time` of the object using the same heuristics as in the Squid proxy [30], which enforces a maximum TTL of 4,320 min (3 days). The average TTL value across all responses is around 3,730 min. Judging from the large number of the 304 requests shown in Figure 9, this limit might be excessive for many sites. Since reducing the limit has a danger of increasing the hit rate at web sites, it would be in content providers' interests to provide explicit expiration times for their responses since doing so would reduce the load on their servers due to 304 requests.

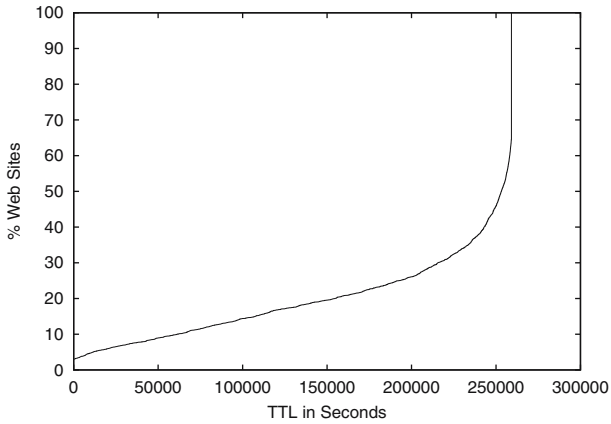


Figure 13 CDF of average TTL for the top 3,000 Web sites.

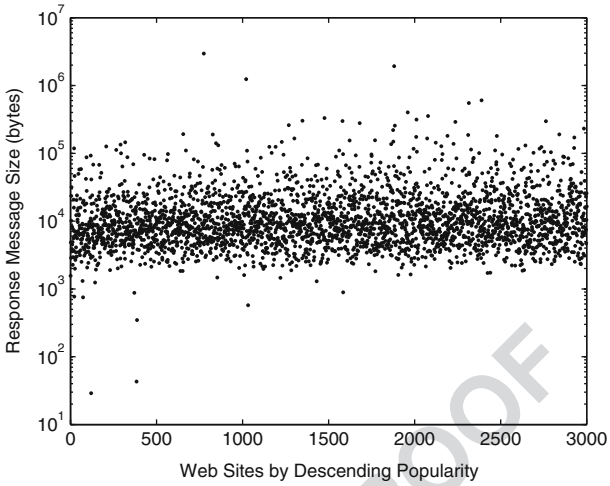


Figure 14 Average response size for the top 3,000 Web sites. The *y*-axis is in log scale.

5.4 Response size, header size, compression

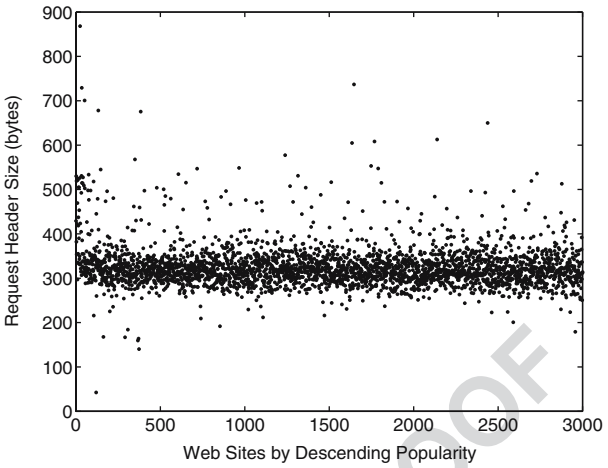
About 62% of response messages specify the `Content – Length` field. Overall, these response messages report an average content length of 8,968 bytes.[2] Figure 14 shows the scatter plot of average response size for the top 3,000 Web sites in descending order of popularity (note that the *y*-axis is in log scale). We find little correlation between the popularity of a Web site and its average message sizes.

Figure 15 shows the header size distribution of request and response messages for the top 3,000 Web sites in descending order of popularity. The majority (90%) of them have an average request header size between 270 bytes and 400 bytes, and an average response header size between 170 and 282 bytes. If every Web site is weighted equally, the average header size is 325 bytes for request messages and 230 bytes for response messages. On the other hand, if we compute the average over the aggregate of all messages, the average size of request headers jumps to 402 bytes, while the average size of response headers remains relatively unchanged at 216 bytes. The largest average request header size we observed in the trace is 868 bytes, which is due to the use of cookies. Comparing Web sites according to their popularity, we see little correlation between the popularity of a Web site and the header sizes of its request and response messages.

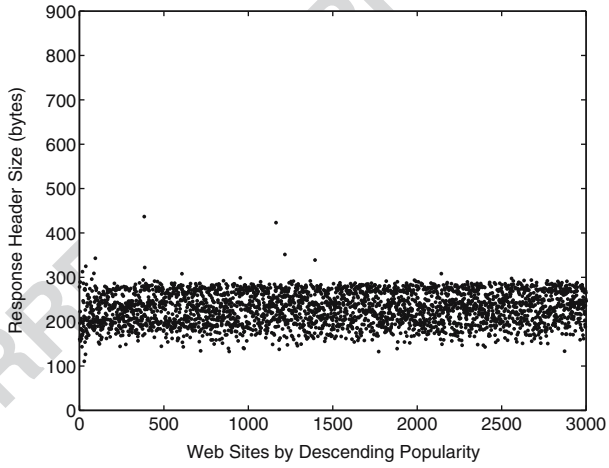
The average request cookie size per Web site is 63 bytes, while across all requests the average Cookie size is 64 bytes. Most Web sites (90%) have an average cookie size between 12 and 130 bytes.

Overall, about 84% of requests indicate the willingness to accept a compressed response. This is consistent with the common impression that compression is widely supported in modern browsers. Those requests that do not support compression likely come from scripts or some out-of-date browsers. Somewhat surprisingly, only 8 out of the total 3,000 Web sites ever return an object with a `Content – Encoding` or `Transfer – Encoding` header indicating some sort of compression. Even these sites returned on average only 5% of their objects compressed.

[2] Note that we cannot compute the length of messages that use Chunk-encoding since our trace only captures the first packet of a response.



(a) Request Header



(b) Response Header

Figure 15 Average header size for the top 3,000 Web sites in descending order of popularity.

6 Estimation of CDN benefits

Content delivery networks (CDNs) deploy caches throughout the Internet to move content closer to the client and decrease client access time. CDNs are also used to decrease hit rate and bandwidth consumption at the Web site. In this section we use trace-driven simulation to study the CDN benefits for Web sites in our trace. Because of recent dramatic increases in disk sizes, we assume that CDN caches have unlimited cache capacity.

Cassandra currently implements a model of a full-time revalidation CDN, which assumes that all user requests are routed through CDN caches, and uses a standard time-to-live (TTL) approach to decide the validity of cached responses. For expired responses, Cassandra's CDN module simulates `If-Modified-Since` requests.

To map clients to CDN caches, we group clients into clusters using a network-aware clustering tool [19]. Since clusters group topologically close clients, we assume

that CDN caches are assigned to clients at the granularity of clusters: all clients in the same cluster are assigned to the same cache. In our simulations, each client cluster is randomly assigned to one of the caches when it generates its first request, and uses that cache afterward. Since we do not measure latency effects in this study, a different cluster-to-cache assignment will not change our results in any significant way.

Given different cache products behave differently with respect to content cacheability, we estimate the lower and upper bounds of the potential benefits of using a CDN on a workload. We produce the upper bound by assuming ideal content cacheability and unlimited lifetimes for cached objects. In other words, we assume that, after the initial request to an object, a CDN cache serves all subsequent requests to the object from its cache. We generate a lower bound by modeling actual content cacheability and consistency: We simulate the effects of existing cookie (assuming that cached objects requested with a cookie are revalidated by the cache as is done by default by NetCache) and cache-controlling headers in the trace and use Squid's default heuristics for expiring cached responses. In practice, CDNs will fall between these bounds. The upper bound is idealistic because it assumes that all content is cacheable and never changes. The lower bound is overly conservative for two reasons. First, in terms of bandwidth consumption, a shared CDN cache can convert some requests (e.g., those with cookies) from unconditional downloads to `If − Modified− Since` requests. For example, if the request from the client carries a cookie, but the response from the server does not depend on the value of the cookie, then the server may return a `304 Not Modified` response instead of the full object. Given the large degree of indiscriminate cookie use we observed in our trace analysis, this is quite likely to happen. Second, some CDN caches may employ more aggressive cookie treatment as discussed in Section 5.1.

These results are useful in two ways. First, the lower and upper bounds indicate the range for potential benefits of using a CDN. Second, the difference between the lower and upper bounds indicate the extent of performance benefits the site could obtain from improving the cacheability of its content (e.g., by making more informed, targeted use of cookies).

We focus on the top 100 Web sites in the following analyzes because these sites are the ones most impacted by the use of a CDN. For the experiments, we make the following modifications to the trace. Our CDN simulations require cacheability information, which is not present in all responses, specifically `304 Not Modified` responses (other responses with no cacheability information are negligible [6]). When a `304 Not Modified` response is the first response to a request for an object, our simulation must discard it, since this response contains no meaningful information for our CDN cache (remember that we are simulating `If − Modified − Since` requests for objects once they are in-cache). Subsequent responses for this same object may contain cacheability information, which we use. The reduction of request rate to the 100 most popular Web sites due to discarding these requests was insignificant.

6.1 Peak request rate

We start by examining the impact on the peak request rates of the Web sites when using a CDN. Reducing the peak request load on origin servers is one of the key benefits of using a CDN. Figures 16 and 17 show the peak request rate of the 100 most popular Web sites with and without using a CDN. Figure 16 shows peak request rates using the actual content cacheability characteristics of the requests and

responses in the trace. Figure 17 shows peak request rates assuming content is ideally cacheable: cached objects have unlimited lifetimes, and CDN caches can serve all subsequent requests to an object. Note that Figure 17 repeats the optimistic CDN simulation from [6], but the leading `304 Not Modified` request/response pairs that appear in the trace before the "real" response with the content are removed.

We compute the peak request rate for each Web site at the granularity of 10 s across the entire trace. Each bar corresponds to a Web site and the Web sites are shown in order of decreasing request popularity. The entire bar shows the peak request rate for that Web site across our trace without using a CDN. The dark part of each bar corresponds to the peak request rate of that Web site when a CDN is used. The white part of each bar shows the portion of the peak request rate handled by the CDN. It shows directly the benefit of using the CDN for that Web site in terms of the reduction in peak request rate.

For the 100 most popular sites shown in Figure 16, a CDN would decrease the peak request rate across all sites by a factor of 1.4. This is averaged over the peak request rate of each individual site, regardless when each site reaches its peak and how much traffic each site contributes. If the sites improved the cacheability of their content, in the ideal case a CDN would decrease the peak request rate across all sites by a factor of 2.5, a 79% improvement.

Another important observation from these graphs is that different sites show drastically different request rate reductions. For example, Figure 16 shows that the most popular site has its peak request rate more than halved (reduced by 52%) when using a CDN, but the second most popular site achieves little benefit (reduced by 1.1%). Since some sites benefit significantly from using a CDN while others do not, individual sites must analyze their specific workload to estimate the benefits of a CDN service. One of the goals of our Cassandra tool [7] is to satisfy this need.

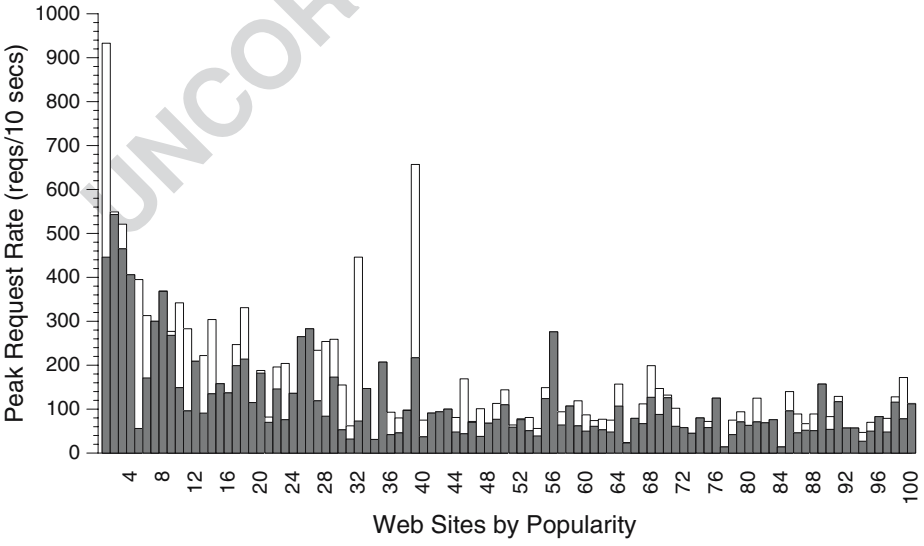


Figure 16 Comparison of peak request rate with a CDN (*dark part of each bar*) and the peak request rate without a CDN (total height of each bar). Models content cacheability and consistency.

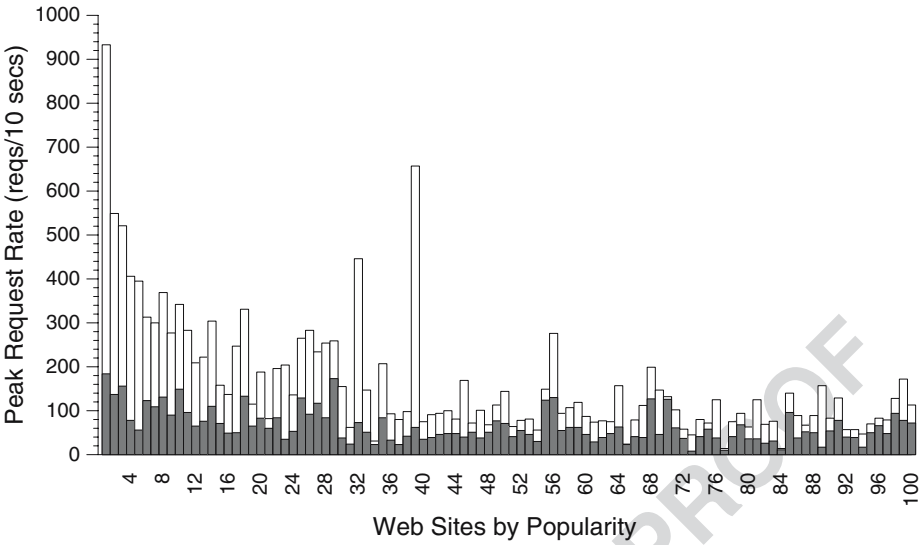


Figure 17 Comparison of peak request rate with a CDN (*dark part of each bar*) and the peak request rate without a CDN (total height of each bar). Assumes ideal cacheability.

Comparing Figures 16 and 17 we see that some sites might increase their CDN benefits significantly by improving the cacheability of their content. For example, for the second most popular site with current cacheability, a CDN decreases the site's peak request rate by only 1.1%, but by improving the cacheability of its content a CDN could decrease the site's peak request rate by 75% in the ideal case (an improvement of 67%). Of course, the ideal case is an upper bound that may be difficult to achieve completely in practice. Nevertheless, a site can use the difference between the realistic and ideal cases to determine whether it is worth the effort to examine and consider the cacheability of its content. In the case of the second most popular site, it appears worthwhile, while for several other sites (such as 70th and 97th most popular sites) it does not seem to be the case.

6.2 Average byte rate

CDNs also reduce the bandwidth requirements for Web sites, in addition to reducing server load by alleviating peak request rates. Reducing bandwidth can directly reduce costs for both individual Web sites and for Web server farms. To evaluate the impact of CDNs on bandwidth requirements, we study the impact of our simulated CDN on the average byte rate of the Web sites in our trace. (Ideally, we would like to evaluate the effect of CDNs on peak byte rate as well; however, since our trace does not include transmission times for responses, we could not compute this metric for this experiment.)

Figures 18 and 19 show the average byte rate for the 100 most popular Web sites in our trace with and without a CDN. Figure 18 shows the average byte rate using the actual content cacheability characteristics of the requests and responses in the trace,

and Figure 19 assumes content is ideally cacheable. Note that Figure 19 repeats the optimistic CDN simulation from [6] but with the leading `304 Not Modified` request/response pairs removed. We compute the average byte rate for a Web site by totaling the header and content lengths of all transactions to the site in the trace, then dividing by the trace duration. The height of each bar shows the average byte rate for the Web site without a CDN. The dark part shows the average byte rate at the Web site when using a CDN, while the white part shows the average byte rate handled by the CDN.

One can consider bandwidth demands from two perspectives—the individual Web sites or the server farm as a whole. For the sites shown in Figure 18, a CDN would decrease total server farm bandwidth demand by a factor of 1.8. These results model the content cacheability and consistency found in the original trace, and represent a lower bound on the benefits of a CDN. In contrast, for the results shown in Figure 19 a CDN would decrease bandwidth consumption by a factor of 3.3. These results model ideal content cacheability, and represent an upper bound. From both graphs, we see that using a CDN for the server farm can significantly reduce bandwidth given its current workload, and that improving the content cacheability of the sites has the potential to make a CDN even more effective for the server farm.

Turning to individual sites, we again see high variability in their CDN benefits. For example, as with the peak request rate metric, the second most popular site achieves little benefit in terms of bandwidth on its current workload (Figure 18), but can potentially gain considerable bandwidth improvements by improving the cacheability of its content (Figure 19). The highly variable amount of bandwidth savings again underscore the need for a Cassandra-style tool for performance analysis and tuning.

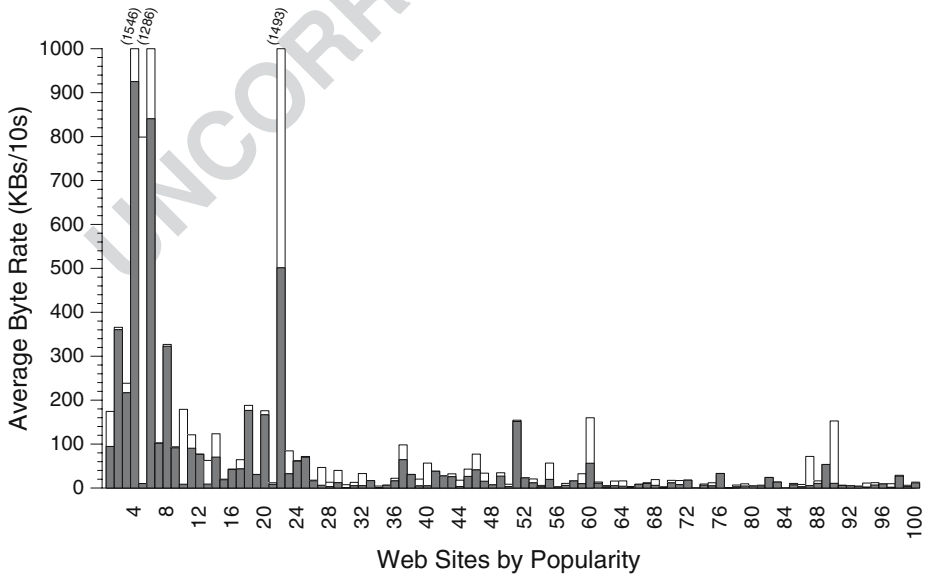


Figure 18 Comparison of average byte rate with a CDN (*the dark part of each bar*) and the average byte rate without a CDN (total height of each bar). Models content cacheability and consistency.

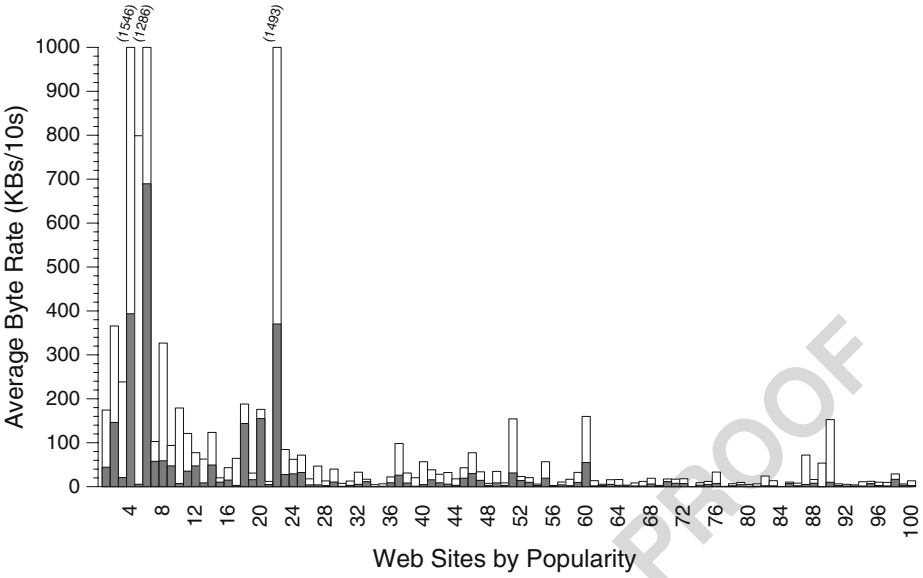


Figure 19 Comparison of average byte rate with a CDN (*the dark part of each bar*) and the average byte rate without a CDN (total height of each bar). Assumes ideal cacheability.

7 Validation of CDN simulation

We validated our CDN benefit analysis by comparing its results with the results observed by running the same trace through a real cache. This section describes our validation methodology and the results, which show virtually identical performance of the simulated and the real testbed environments. To our knowledge, ours is the first validation study of a simulator with detailed cacheability and revalidation models.

7.1 Methodology

Figure 20 shows our validation testbed architecture. It contains a workload generator driven by the requests from the original trace, a real cache in the middle, and the server stub, which supplies responses of the proper size and with proper HTTP headers taken from the trace. We validate our CDN simulation by replaying the original trace through this architecture, and comparing the observed metrics with the same metrics obtained from running the same trace through our CDN simulator. In particular, we validate the request rate at the origin, both over the entire trace and over the peak 10-s period. The comparison of these metrics is indicative of the simulation accuracy, and they are straightforward to compute based on proxy cache statistics.

We implemented the workload generator in Perl using the libwww toolkit [23] and the high resolution timing package HiRes [15]. For the server stub, we used the Apache HTTP server [1], since it supports returning objects with individually predefined headers. Finally, we used the Squid proxy [30] with most of the default settings as the cache.

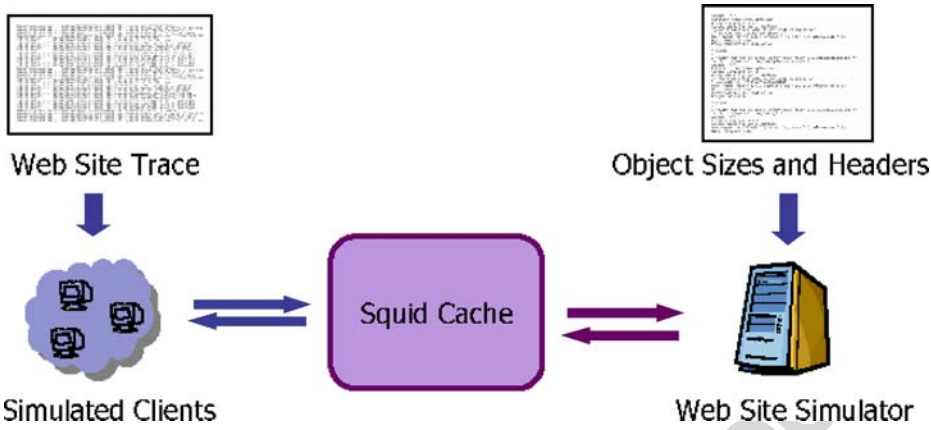


Figure 20 Validation architecture.

To emulate the actual execution to the greatest extent, we set the entire testbed platform back in time of the start of the trace, and replay each request at the time of its appearance in the trace. In other words, the testbed experiments run in shifted real time, and a 26-h trace takes 26 h to replay. To avoid clock synchronization issues, we run the entire testbed—the workload generator, the cache, and the origin server stub—on the same machine, a 2.66 GHz Pentium 4 with 504 MB of memory running Red Hat Linux 3.3.3-7. We verified that the machine was never overloaded during the experiments, so our trace timing properties were preserved.

We populate our stub server with "dummy" objects, with the length and HTTP headers taken from the trace. We store objects in the same directory hierarchy as referenced in the original URLs and we generally access the object with the same URL as in the original trace, except that characters that cannot be easily included in filenames are stripped out. In particular, CGI objects are stored on disk without the tell-tale "?" in the filename because Linux treats "?" as a special character. However, we ensure that this change does not affect our validation experiment by retaining the "?" in the URLs used to access these objects. In this way, the cache will see the requests to these objects as CGI requests while the server stub will truncate these URLs at the "?" and return the corresponding objects along with their original HTTP headers. The fact that the server stored these objects statically rather than computing them on the fly has no effect on the cache behavior and hence on the CDN benefits.

Because we focus on the request rates in our validation experiment, we do not attempt to recreate object changes in the trace, but assign each object its initial size in the trace. A study considering bandwidth or cache consistency could take object changes into account as well.

We configure the Squid cache using the default settings with two exceptions. First, because we are comparing against an infinite-sized cache in the simulations, we expand disk space allocated to Squid to accommodate all objects. Second, Squid usually caches `404NotFound` responses for a fixed period of time. Since our simulation does not cache these responses, we configure Squid not to cache them either.

Table 7 Web sites selected for validation.

Web site	Hours	Number of requests	Unique properties
Commercial	26 h	1.4 M	Large, commercial trace
Commercial w/ Cookies	26 h	43 K	Heavily cookied
Educational/ Research	24 h	83 K	Few server-side headers, many client-side headers

7.2 Validated web sites

We selected three Web sites from our trace for our validation experiment. Table 7 lists the properties of these sites. We refer to these Web sites as `Commercial`, `Commercial with Cookies`, and `Educational/Research`. The first Web site contains a very high request rate (1.4M) but few cookies. The second Web site represents a much lighter commercial workload with a heavy use of cookies (43 K requests, with 95% of requests having a `Cookie` header and 0.6% of responses having a `Set − Cookie` header). The final site is an educational/research site that uses very few server-side cache-controlling headers, but contains a relatively large (24%) number of client-side cache-controlling headers. These Web sites represent a spectrum of behaviors and exercise the CDN simulator under a variety of scenarios: server-side cache-controlling headers, cookie headers, client-side cache-controlling headers, heavy workloads, lighter workloads, commercial workloads and non-commercial workloads.

7.3 Validation of cacheability model

Table 8 summarizes validation results for the total number of requests at the origin site. It shows in columns 3–5 the simulated optimistic and conservative bounds for this metric as well as the actual number of requests obtained from replaying the trace through Squid. For the two commercial sites, the actual number lies indeed within the simulated bounds, while for the educational site it just slightly exceeds the conservative estimation. Although the actual number of requests exceeded the conservative simulated value by only 49 requests out of 27 K total, we investigated

Table 8 Validation of CDN simulation: total requests.

Web site	Total requests in trace	Requests at origin server (% improvement)			
		Simulated optimistic	Simulated conservative	Actual squid	Simulated squid
Commercial	1,420,267	1,418 (>99%)	55,782 (96%)	55,697 (96%)	55,782 (96%)
Commercial w/ cookies	42,635	253 (99%)	42,502 (0%)	10,940 (74%)	10,766 (75%)
Educational/ research	82,749	14,876 (82%)	26,848 (68%)	27,099 (67%)	26,848 (68%)

the reason for this discrepancy. The discrepancy turned out to be due to two factors, that Squid does not cache objects with `robots.txt` in the URL and it does not cache objects over 4 MB in size. Our simulation assumed both these object kinds are cacheable, and so satisfied slightly more requests. The minuscule number of the affected requests notwithstanding, this underscores the importance of validating simulation results.

Another question from Table 8 is whether the fact that range includes the actual performance is due to the accuracy of the simulation or simply the artifact of the wide range between the conservative and optimistic simulation numbers. To judge how accurately we account for all the factors in cache behaviors, we modified the simulator to model the Squid behavior, in addition to the optimistic and conservative behaviors. Specifically, the modified simulator does not revalidate objects requested with cookies and in fact ignores cookies in both requests and responses altogether. The last column in Table 8 shows the resulting number of requests at the origin. It obviously does not change for non-cookied sites but becomes virtually identical to Squid in all three sites. The actual value slightly exceeds the simulated value for the same reasons mentioned above (the `robots.txt` objects and objects over 4 MB).

Let us now turn to the peak request rates at the origin server. Table 9 shows the validation results for this metric. The Table lists the peak requests over 10 s for each Web site. For the validation (The "Actual Squid" column), we compute the peak request rate at origin in the following way. Since the actual timed replay is subject to variability, we compute the peaks based on the timing information in the original trace. In other words, we use Squid logs to track which objects hit in the cache and which missed. Then, we use the original trace timing information to compute the 10-s bins over which the peak request rate (i.e., misses in the Squid cache) is calculated. Thus, the peaks are computed as if the replay happened on the original time scale, with Squid processing delays factored out. Because Squid delays are negligible relative to request inter-arrival times, ignoring them makes the results deterministic without affecting any of the conclusions.

Like Tables 8 and 9 shows a vary close match between simulated and actual Squid performance. Further, unlike Table 8 the actual performance always falls in-between the conservative and optimistic simulated values. The reason is that the small number of extra misses in actual Squid due to requests for large objects and `robots.txt` did not occur during peak 10 s intervals and hence did not affect the results.

Overall, the validation of the two metrics shows high accuracy of our simulator in all three sites considered.

Table 9 Validation of CDN simulation: peak request rates.

Web site	Peak requests (Req/10s) in trace	Peak requests at origin server (req/10 s) (% improvement)			
		Simulated optimistic	Simulated conservative	Actual squid	Simulated squid
Commercial	960	81 (92%)	89 (91%)	87 (91%)	89 (91%)
Commercial w/ Cookies	146	39 (73%)	146 (0%)	81 (45%)	80 (45%)
Educational/ Research	174	57 (67%)	62 (64%)	62 (64%)	62 (64%)

7.4 Validation of shared cache model

Finally, during validation replay all of our requests are generated from one client, while our original trace contains requests from many clients. Our simulator does not distinguish between requests from the same or different clients given the negligible number of `cache – control : private` headers in the trace. However, we are not guaranteed that a real cache would behave the same. For example, a cache might ignore cookies in requests from the same client but revalidate objects requested with different cookies from different clients. To validate our simulated shared cache model, we must determine whether Squid treats requests differently depending on client IP addresses. Consequently, we conducted the following validation experiment.

We divided our heavily cookied trace into two subtraces by client IP address, so that requests from the same IP address always wend into the same subtrace. We verified that the two subtraces exhibit a large degree of sharing, with 235 of 258 total URLs references in both subtraces. Further, only a small number of these shared URLs were requested with the same cookie in both subtraces (only 2 shared URL/cookie combinations of the 22,993 in total), indicating that the same objects are being requested with different cookies in the two subtraces. We then replayed both traces simultaneously at trace time from two client processes. If Squid were to treat requests from different clients differently, and in particular if it were to revalidate cookied objects from different clients (as opposed to ignoring cookies as we observed in previous experiments), requesting the objects from the two subtraces would show more cache misses than from one trace. Instead, we found that the miss rate was exactly the same during the two-trace replay as during the one-trace replay. We conclude that our shared cache model is justified, and it makes no difference whether we request the objects from one client or many clients during the trace replay in the validation experiments.

8 Conclusion

Understanding the properties of Web sites is important for content delivery. This paper has investigated the characteristics of the 3,000 busiest Web sites in a large server farm. We found that the Web server farm workload contains a much higher degree (66%) of uncacheable responses and responses that require mandatory cache validations. We found that many of the sites use cookies indiscriminately and fail to utilize the full cache-controlling features provided by the HTTP/1.1 standard. This results in suboptimal caching behavior for content delivery. We have also analyzed the benefit of content delivery networks using trace driven simulation. To investigate the potential impact of content cacheability, we analyzed these benefits under both idealized and conservative cacheability assumptions. Further, we validated our analysis with a real cache deployment driven from the same trace used in the simulation. We found that CDNs can achieve significant reduction in bandwidth and request rate at the origin servers. However, content cacheability has a significant impact on the benefits, and the benefits vary dramatically for different sites.

There are several directions for future work. We plan to extend our study to a larger time scale and analyze the dynamics of Web sites with respect to time. For example, it will be interesting to see how the traffic changes during different times

of the day, during different days of the week, and the correlation between past and future traffic. We also plan to enhance our simulation to estimate the benefits of so-called overflow CDNs (i.e., CDNs used by a Web site only during periods of peak demand) and also model other content delivery technologies such as prefetching.

Finally, although numerous techniques have been developed, implemented, and standardized for improving the performance of Web content delivery, we find that most sites either do not take advantage of such techniques (e.g., cache control directives) or unknowingly inhibit them (e.g., indiscriminate cookie use). This situation arises because of the complexity of contemporary Web site content, the complexity of the techniques for improving content delivery, and the difficulty in measuring and evaluating the effectiveness of content delivery optimizations on Web sites. Without the ability to quantify the effect of their content delivery decisions, it is difficult for Web site maintainers to understand the implications of their decisions. We believe one promising approach for addressing this situation is to develop a tool that gives Web site developers more insight into how their site performs and interacts with advanced content delivery mechanisms.

Acknowledgments We are grateful to Oliver Spatscheck for stimulating discussions and for answering our numerous questions. We also thank the anonymous reviewers for their helpful comments. Support for this work was provided in part by AT&T support of the UCSD Center for Networked Systems. Voelker was supported in part by AFOSR MURI Contract F49620-02-1-0233.

References

1. The Apache software foundation. http://www.apache.org
2. Arlitt, M., Jin, T.: Workload characterization of the 1998 World Cup Web site. Technical Report HPL-1999-35R1, HP Labs, October 1999
3. Arlitt, M.F., Williamson, C.L.: Web server workload characterization: the search for invariants. In: Proc. of ACM SIGMETRICS, pp. 126–137 (1996)
4. Arlitt, M., Friedrich, R., Jin, T.: Workload characterization of a Web proxy in a cable modem environment. Technical Report HPL-1999-48, Hewlett Packard Labs, April 1999
5. Barford, P., Bestavros, A., Bradley, A., Crovella, M.: Changes in Web client access patterns: characteristics and caching implications. World Wide Web **2**, 15–28 (1999)
6. Bent, L., Rabinovich, M., Voelker, G.M., Xiao, Z.: Characterization of a large web site population with implications for content delivery. In: Proc. of the World Wide Web Conference, May 2004
7. Bent, L., Rabinovich, M., Voelker, G.M., Xiao, Z.: Towards informed web content delivery. In: Proc. of the Ninth International Workshop on Web Content Caching and Distribution (WCW'04), October 2004
8. Brewington, B.E., Cybenko, G.: How dynamic is the Web? In: Proc. of the 9th Int. World Wide Web Conference (2000)
9. Cherkasova, L., Karlsson, M.: Dynamics and evolution of Web sites: Analysis, metrics and design issues. Technical Report HPL-2001-1R1, Hewlett Packard Laboratories, 16 July 2001
10. Cranor, C., Johnson, T., Spatscheck, O.: Gigascope: a stream database for network applications. In: Proc. of ACM SIGMOD, June 2003
11. Douglis, F., Feldmann, A., Krishnamurthy, B., Mogul, J.: Rate of change and other metrics: a live study of the World Wide Web. In: Proc. of the USENIX Symp. on Internet Technologies and Systems, pp. 147–158, December 1997
12. Duska, B., Marwood, D., Feeley, M.J.: The measured access characteristics of World Wide Web client proxy caches. In: Proc. of the First USENIX Symp. on Internet Technologies and Systems, pp. 23–36, December 1997

13. Feldmann, A., Cáceres, R., Douglis, F., Glass, G., Rabinovich, M.: Performance of Web proxy caching in heterogeneous bandwidth environments. In: Proc. of IEEE INFOCOM, pp. 107–116 (1999)
14. Gribble, S.D., Brewer, E.A.: System design issues for Internet middleware services: deductions from a large client trace. In: Proc. of the First USENIX Symp. on Internet Technologies and Systems, pp. 207–218, December 1997
15. The HiRes Timing Library. http://www.search.cpan.org/~jhi/Time-HiRes-1.66/HiRes.pm
16. Iyengar, A.K., Squillante, M.S., Zhang, L.: Analysis and characterization of large-scale Web server access patterns and performance. World Wide Web **2**(1–2), 85–100, June (1999)
17. Jung, Y., Krishnamurthy, B., Rabinovich, M.: Flash crowds and denial of service attacks: characterization and implications for CDNs and web sites. In: Proc. of the 11th Int. World Wide Web Conference, May 2002
18. Kelly, T.: Thin-client Web access patterns: measurements from a cache-busting proxy. In: Proc. of the Int. Workshop on Web Content Caching and Distribution (2001)
19. Krishnamurthy, B., Wang, J.: On network-aware clustering of Web clients. In: Proc. of ACM SIGCOMM, August 2000
20. Krishnamurthy, B., Wills, C.E.: Analyzing factors that influence end-to-end Web performance. Comput. Networks **33**(1–6), 17–32 (2000)
21. Krishnamurthy, B., Arlitt, M.: PRO-COW: Protocol compliance on the Web: a longitudinal study. In: Proc. of the 3rd USENIX Symp. on Internet Technologies and Systems, pp. 109–122 (2001)
22. Krishnamurthy, B., Wills, C., Zhang, Y.: On the use and performance of content distribution networks. In: Proc. of the First ACM SIGCOMM Internet Measurement Workshop, pp. 169–182, November 2001
23. libwww: http://search.cpan.org/~gaas/libwwwperl5.803/
24. Manley, S., Seltzer, M.: Web facts and fantasy. In: Proc. of the USENIX Symp. on Internet Technologies and Systems, pp. 125–133, December 1997
25. Mogul, J.C.: Network behavior of a busy Web server and its clients. Technical Report 95/5, Compaq Western Research Lab, October 1995
26. Mogul, J.C., Douglis, F., Feldmann, A., Krishnamurthy, B.: Potential benefits of delta encoding and data compression for HTTP. In: Proc. of ACM SIGCOMM, pp. 181–194 (1997)
27. Padmanabhan, V.N., Qiu, L.: The content and access dynamics of a busy Web site: findings and implications. In: Proc. of ACM SIGCOMM, August 2000
28. Pitkow, J.E.: Summary of WWW characterizations. World Wide Web **2**, 3–13, June (1999)
29. Raunak, M.S., Shenoy, P.J., Goyal, P., Ramamritham, K.: Implications of proxy caching for provisioning networks and servers. In: Proc. of ACM SIGMETRICS, pp. 66–77 (2000)
30. The squid Web proxy cache. version 2.5. http://www.squid-cache.org
31. Wills, C.E., Mikhailov, M.: Examining the cacheability of user-requested Web resources. In: Proc. of the Fourth Int. Workshop on Web Content Caching and Distribution, April 1999
32. Wolman, A., Voelker, G.M., Sharma, N., Cardwell, N., Brown, M., Landray, T., Pinnel, D., Karlin, A., Levy, H.: Organization-based analysis of Web-object sharing and caching. In: Proc. of the USENIX Symp. on Internet Technologies and Systems (1999)
33. Wolman, A., Voelker, G.M., Sharma, N., Cardwell, N., Karlin, A., Levy, H.M.: On the scale and performance of cooperative Web proxy caching. In: Proc. of ACM SOSP, pp. 16–31, December 1999