Towards Understanding Modern Web Traffic

Sunghwan Ihm†
Department of Computer Science
Princeton University
sihm@cs.princeton.edu

Vivek S. Pai
Department of Computer Science
Princeton University
vivek@cs.princeton.edu

ABSTRACT

As Web sites move from relatively static displays of simple pages to rich media applications with heavy client-side interaction, the nature of the resulting Web traffic changes as well. Understanding this change is necessary in order to improve response time, evaluate caching effectiveness, and design intermediary systems, such as firewalls, security analyzers, and reporting/management systems. Unfortunately, we have little understanding of the underlying nature of today's Web traffic.

In this paper, we analyze five years (2006-2010) of real Web traffic from a globally-distributed proxy system, which captures the browsing behavior of over 70,000 daily users from 187 countries. Using this data set, we examine major changes in Web traffic characteristics that occurred during this period. We also present a new Web page analysis algorithm that is better suited for modern Web page interactions by grouping requests into streams and exploiting the structure of the pages. Using this algorithm, we analyze various aspects of page-level changes, and characterize modern Web pages. Finally, we investigate the redundancy of this traffic, using both traditional object-level caching as well as content-based approaches.

Categories and Subject Descriptors

C.2.m [**Computer-Communication Networks**]: Miscellaneous

General Terms

Measurement, Design, Performance

Keywords

Web Traffic Analysis, Web Caching, Content-based Caching

1. INTRODUCTION

The World Wide Web is one of the most popular Internet applications, and its traffic volume is increasing and evolving due to the popularity of social networking, file hosting, and video streaming sites [29]. These changes and growth of Web traffic are expected to continue, not only as the Web becomes a *de facto* front-end for many emerging cloud-based services [47], but also as applications get migrated to the Web [34].

Understanding these changes is important for overall system design. For example, analyzing end-user browsing behavior can lead to a Web traffic model, which in turn can be used to generate a synthetic workload for benchmarking or simulation. In addition, analyzing the redundancy and effectiveness of caching could shape the design of Web servers, proxies, and browsers to improve response times. In particular, since content-based caching approaches [28, 49, 50] are a promising alternative to traditional HTTP object-based caching, understanding their implications for Web traffic and resource requirements (*e.g.*, cache storage size) could help reduce bandwidth and improve user experience.

While much research activity occurred a decade ago aimed at better understanding the nature of Web traffic [9, 11, 35, 56, 63], it subsided just as the Web changed significantly, and we must therefore update our understanding of today's Web traffic. However, there are several challenges. First, examining changes over time requires large-scale data sets spanning a multi-year period, collected under the same conditions. Second, earlier Web page analysis techniques developed for static pages are not suitable for modern Web traffic that involves dynamic client-side interactions (*e.g.*, Ajax [18]). Third, understanding the effectiveness of content-based caching approaches requires full content data rather than just access logs.

In this paper, we analyze five years (2006-2010) of real Web traffic from a globally-distributed proxy system, which captures the browsing behavior of over 70,000 daily users from 187 countries. Using this data set, we examine major changes in Web traffic characteristics that occurred during this period. We also present a new Web page analysis algorithm that is better suited for modern Web page interactions by grouping requests into streams and exploiting the structure of the pages. Using this algorithm, we analyze various aspects of page-level changes, and characterize modern Web pages. Finally, we investigate the redundancy of this traffic, using both traditional object-level caching as well as content-based approaches.

Our contributions and key findings are the following:

High-Level Characteristics The rise of Ajax and video content has impacted a number of different traffic measures. Ajax has caused in increase in the sizes of JavaScript and CSS [19] objects, and browsers have increased their simultaneous connection limit to better support it, resulting in burstier traffic but also improved client latency. Flash video (FLV) has grown to dominate video traffic, pushing the share of other video formats lower, and also in-

†Current affiliation: Google Inc.

IMC'11, November 2–4, 2011, Berlin, Germany.

creasing bandwidth consumption. Ajax and JavaScript are heavily used in user tracking and we find that analytics sites are reaching an ever-widening fraction of Web users, with some sites being able to track as much as 65% of our client population, which may have privacy implications. In addition, we observe clear regional differences in client bandwidth, browser popularity, and dominant content types that need to be considered when designing and deploying systems. Finally, we observe an increase in the number of computers per household over the years in Network Address Translation (NAT) [59] usage, which is likely related to the scarcity of IPv4 addresses.

Page-Level Characteristics We have developed a new Web page analysis algorithm called StreamStructure, and demonstrate that it is more accurate than previous approaches. Using this algorithm, we find that almost half the traffic now occurs not as a result of initial page loads, but as a result of client-side interactions after the initial page load. Also, the pages have become increasingly complex in that both the size and number of embedded objects have increased. Despite this increase, the page loading latency dropped in 2009 and 2010 due to the increased number of simultaneous connections in browsers and improved caching behavior of Web sites. Furthermore, we quantify the potential reduction of page loading latency from various tuning approaches, such as increasing the number of concurrent connections, and prefetching/caching, via simulations. Finally, we present a simple characterization of modern Web pages.

Redundancy and Caching We find two interesting trends in URL popularity: 1) the popular URLs get more popular, and therefore potentially improves caching, but 2) the long tail of the content is also growing, and therefore potentially hurts caching. Also, we find that content-based caching yields 1.8-2.5x larger byte hit rates than object-based caching, and much larger caches can be effectively exploited using intelligent content-based caching to yield nearly ideal byte hit rates. Most of the additional savings of content-based caching are due to partial content overlap – the redundancy across different versions of an object as well as redundancy across different objects. Finally, a small number of aborted requests (1.8-3.1%), mostly video, can negatively impact object-based caching performance because of its huge volume (12.4-30.8%). Worse, their volume would comprise a significant portion of all traffic (69.9-88.8%) if they were fully downloaded.

The rest of this paper is organized as follows: in Section 2, we describe the details of our data set. Section 3 examines the major changes in high-level characteristics of Web traffic. Section 4 presents the detailed page-level analysis with our new Web page analysis technique, and Section 5 analyzes redundancy and caching. Finally, we discuss related work in Section 6, and conclude in Section 7.

2. DATA SET

Data Collection We use traffic from the CoDeeN content distribution network (CDN) [62], a semi-open globally distributed proxy that has been running since 2003, and serves over 30 million requests per day from more than 500 PlanetLab [45] nodes. The term "semi-open" means that while anyone can use CoDeeN by configuring his or her browser, it only allows GET requests from the general public, and limits other methods such as CONNECT, PUT, or POST to only university-based users. When needed, the system redirects user requests to other proxy nodes based on the load and latency. Some requests are cache misses or uncacheable, and need to be retrieved from the origin Web servers. CoDeeN also deploys an automatic robot detection mechanism and has rejected accesses from malicious robots since 2006 [44].

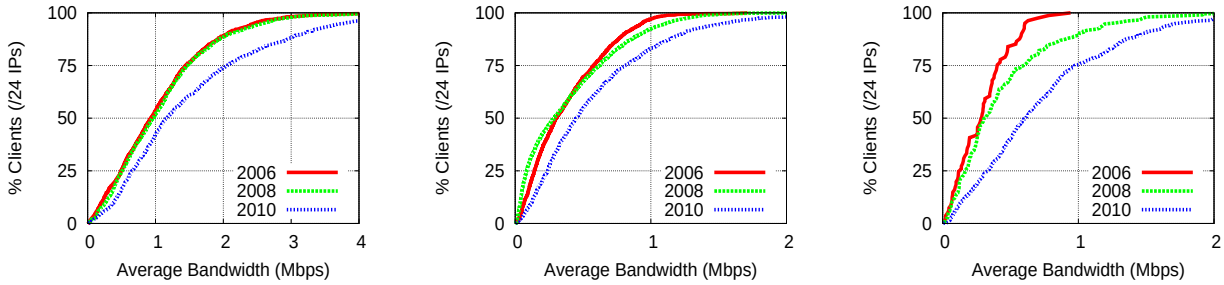
Country		Year				
		2006	2007	2008	2009	2010
USA	Requests (M)	33.5	40.3	24.5	23.2	14.4
	Volume (GB)	391.2	627.2	338.2	316.2	261.5
	# IPs (K)	19.1	21.8	13.6	13.3	12.9
	# Users (K)	23.3	27.0	17.6	16.9	16.7
China	Requests (M)	22.5	88.8	29.9	38.1	22.9
	Volume (GB)	394.5	1,177.8	405.0	409.6	278.4
	# IPs (K)	49.3	94.9	38.8	43.2	33.4
	# Users (K)	53.9	109.7	45.1	51.8	41.9
France	Requests (M)	2.2	3.9	3.3	3.6	3.3
	Volume (GB)	21.6	45.8	33.6	42.9	50.5
	# IPs (K)	3.6	5.1	3.2	3.7	5.1
	# Users (K)	3.9	5.5	3.5	4.3	6.0
Brazil	Requests (M)	1.5	4.5	2.0	3.9	7.1
	Volume (GB)	16.2	54.8	22.8	44.1	100.2
	# IPs (K)	1.4	8.6	3.3	3.1	9.5
	# Users (K)	1.6	10.0	3.8	3.6	10.9
Total	Requests (M)	59.6	137.5	59.7	68.8	47.7
	Volume (GB)	823.5	1,905.6	799.6	812.8	690.6
	# IPs (K)	73.5	130.4	58.9	63.3	61.0
	# Users (K)	82.8	152.2	70.0	76.7	75.5

Table 1: Summary statistics for captured access logs, sampled one month (April) per year.

Our data set consists of two parts. First, CoDeeN records all requests not served from the client's browser cache in an extended W3C log format with timestamp, service time, request URL, method, user agent, content type, referer, response code, and response size. We use these access logs for examining any longitudinal changes in Section 3, 4, and 5. In addition, we capture the full content of the cache-miss traffic between the CoDeeN nodes and the origin Web servers. Using this full content data, we evaluate both object-based and content-based caching approaches, and analyze aborted transfers in Section 5.

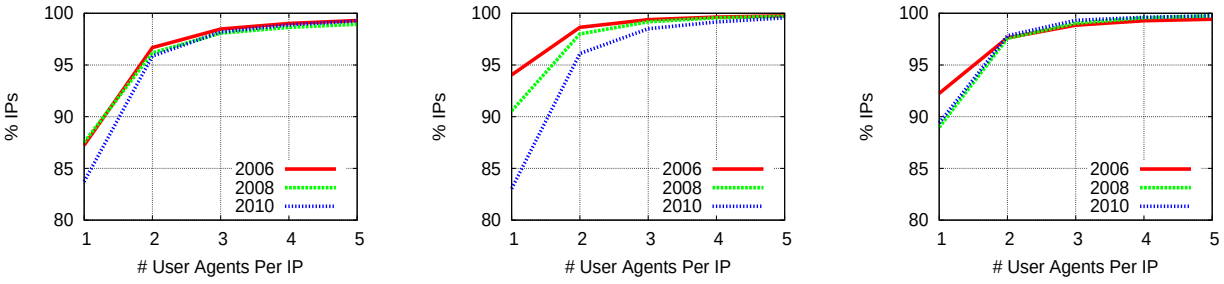
For this study, we consider the data from the five-year period from 2006 to 2010. Due to the large volume of requests, we sample one month (April) of data per year. We only capture the full traffic content in April 2010, and use only traffic logs in all other years. After discarding non-human traffic, the total traffic volume ranges from 3.3 to 6.6 TB per month, and it consists of about 280-460 million requests from 240-360 thousand unique client IPs. The number of users (unique client IP and browser user-agent string pairs) ranges from 280 to 430 thousand, slightly larger than the number of client IPs. The clients IPs originate from 168-187 countries and regions as determined using the MaxMind database [38], and cover 40-60% of /8 networks, and 7-24% of /16 networks. The total number of unique origin servers ranges from 820 thousand to 1.2 million.

We focus on the traffic of users from four countries from different continents – the United States (US), Brazil (BR), China (CN), and France (FR). This essentially generates multiple data sets from different client organizations, and analyzing geographically dispersed client organizations enables us to discover common characteristics of Web traffic across different regions as well as region-specific characteristics. Table 1 shows summary statistics for these countries. In general, the United States and China have larger data sets than France and Brazil, mainly due to their larger client population. The yearly fluctuation of traffic volume is due to the variation of the number of available proxy nodes. Overall, our analysis of four countries covers 48-138 million requests, 691-1906 GB traffic, and 70-152 thousand users per month.



(a) US (b) CN (c) BR

Figure 1: Average client bandwidth: Client bandwidth gets improved over time.



(a) US (b) CN (c) BR

Figure 2: NAT usage: Most (83-94%) client IPs have only one user agent. The number gets slightly bigger over time due to an increase in the number of computers per household.

Users and Content Our large-scale data set spanning many years is much larger than the sets used in previous Web traffic research, and has broader coverage than much previous work, which has typically been restricted to users from a particular university or company [6, 63]. While some self-selection is unavoidable since we are measuring users of our own system, we have found that our user population is quite diverse, covering people who want faster Web access, unfiltered content, better routing, better privacy, and other reasons.

As a high-level check on representativeness, we examine the users of CoDeeN by looking at the `User-Agent` field in the access log, and find that our global trends and regional variations in browser and operating system usage are consistent with other studies [60, 61]. Overall, Firefox and Microsoft Internet Explorer (MSIE) account for more than 86% of the browsers in use over the course of five years in all of the four countries, and more than 83% of the users' operating systems are Windows. In some countries, we see a slightly higher share of Firefox than reported in other studies, which we attribute to the existence of many Firefox Web proxy addons that can use CoDeeN [23]. In most countries, we also observe a decreasing share of MSIE due to the increasing share of other browsers, such as Firefox and Chrome, with the exception of China, which continues to show higher usage of MSIE and Windows.

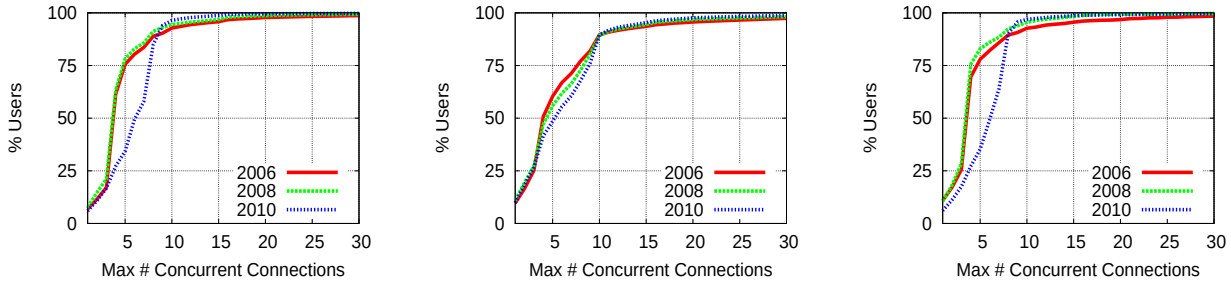
In addition, we investigate content that the users of CoDeeN browse by examining the top sites in terms of the number of requests. The results also correspond to other studies [5]. We observe that globally-popular Web sites, such as Google, YouTube, Yahoo, and Facebook, are commonly ranked high in all of the four countries. Furthermore, locally-popular Web sites also appear high in the ranking. For example, these sites include `craigslist.org`, `go.com`, and `espn.com` in the United States, `baidu.com`, `qq.com`, and `sina.com.cn` in China, `lequipe.fr`, `free.fr`, and `over-blog.com` in France, and `globo.com`, `uol.com.br`, and `orkut.com` in Brazil.

3. HIGH-LEVEL CHARACTERISTICS

In this section, we analyze high-level characteristics of our Web traffic data, looking at the properties of clients, objects, and Web sites.

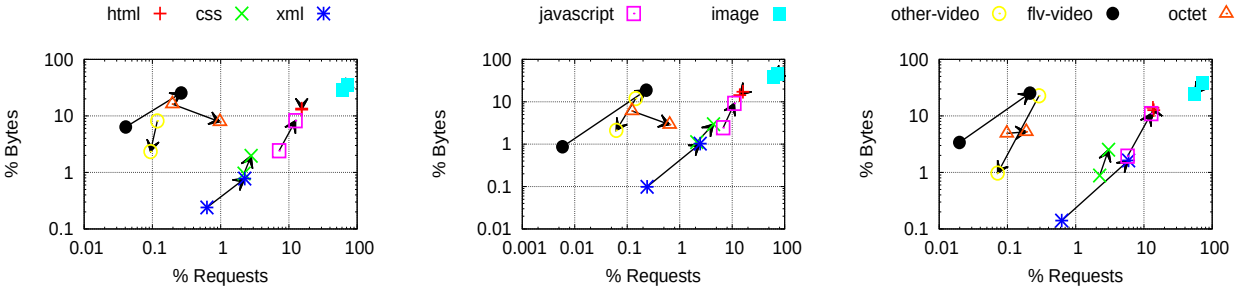
Connection Speed We estimate the client bandwidth by observing the download time for objects – from the time when CoDeeN proxy node receives the request from a client to the time when the proxy finishes sending the object to the client. To minimize the effect of link latency, we consider only those objects that are larger than 1 MB. Figure 1 shows CDFs of average client bandwidth per aggregated /24 IP address, but similar patterns exist when using the 95th percentile of download time. Overall, we observe that the client bandwidth is consistently increasing over time, despite no significant change in PlanetLab's own connectivity in that period. Geographically, the speed of the United States and France is faster than Brazil and China. This scarcity of bandwidth is particularly apparent in 2006, when we did not see any clients in Brazil and China with download speeds exceeding 2 Mbps. Interestingly, there still exist many slow clients with less than 256 Kbps, even in the developed countries.

NAT Usage We analyze the use of Network Address Translation (NAT) in Figure 2 where we present CDFs of the number of different user agents per client IP address. The result shows that while most (83-94%) client IPs have just one user agent, the number gets slightly bigger over time. We attribute this increase to an increase in the number of computers per household over the years, which implies the scarcity of IPv4 addresses. The maximum number of user agents per IP we observe is 69 in the United States, 500 in China and 36 in Brazil. Our estimated number of NATs that have two or more distinct hosts (6-17%) is lower than the numbers from other studies that use client-side measurements or IP TTLs [15, 33, 37], possibly due to methodology differences or from only some of the hosts behind a NAT using the CoDeeN system.



(a) US (b) CN (c) BR

Figure 3: Maximum number of concurrent connections per user: We observe quite a big increase in 2010 due to the browsers increasing the number of connections.



(a) US (b) FR (c) BR

Figure 4: Content type distribution changes from 2006 to 2010: We observe growth of Flash video, JavaScript, CSS, and XML. Images are still dominating request traffic.

Maximum Concurrent Connections Figure 3 shows CDFs of the maximum number of concurrent connections per user agent. We observe quite a big increase in 2010 – the median number grows from 4-5 in 2006 and 2008 to 6-7 in 2010. This increase is mainly because the browsers change the default number of maximum simultaneous connections per server from 4 to 6 in starting in 2008 [4, 51], largely to accommodate Ajax which usually requires many simultaneous connections to reduce latency. In fact, the default number specified in HTTP/1.1 is only 2 [22].

Content Type We observe a shift from static image-oriented Web pages to dynamic rich media Web pages in Figure 4. It presents the content type distribution changes from 2006 to 2010, connected by arrows. The X axis is the percentage of requests, and the Y axis is the percentage of bytes, both in log-scale.

First, we observe a sharp increase of JavaScript, CSS, and XML, primarily due to the popular use of Ajax. We also find a sharp increase of Flash video (FLV) traffic, taking about 25% of total traffic both in the United States and Brazil in 2010, as it eats into the share of other video formats. In addition, while the byte percentage of octet-stream traffic sees a general decrease, its percentage of requests actually increases. This may be related to the custom use of HTTP as a transport protocol for exchanging binary data in many applications. Still, image traffic, including all of its subtypes, consumes the most bandwidth.

Despite the growth of embedded images in Web pages, we do not see a corresponding surge in their numbers in the traffic patterns. We believe that this is due to the improved caching behavior of many Web sites that separate the cacheable parts of their content on different servers and use long expiration dates. As a result, most of these images are served from the browser cache after the initial visit to the Web site.

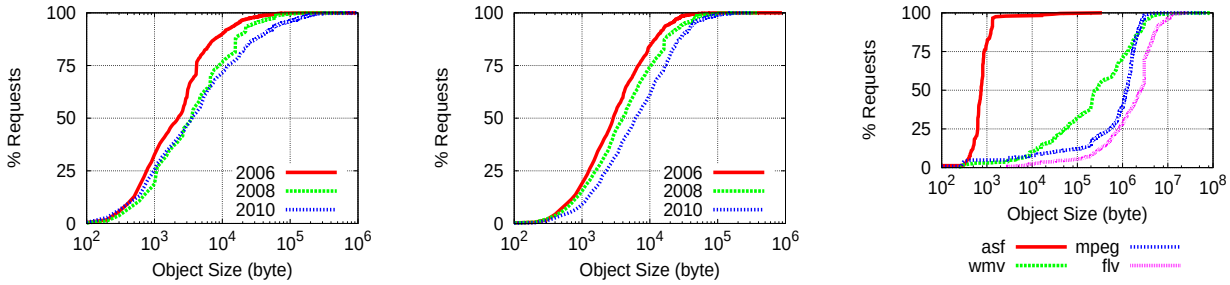
Object Size We find that the size of JavaScript and CSS to be increasing steadily over time. As an example, Figure 5 (a) presents CDFs of JavaScript sizes in France, and we show CDFs of CSS sizes in China in Figure 5 (b), from 2006 to 2010. We omit the similar results of other countries due to space constraints. The increased code size of JavaScript and advanced CSS is likely related to the increasing popularity of Ajax. In general, other content types do not show consistent size changes over time.

While there seems to be no significant size changes over time in video objects, we observe FLV objects are bigger than other video in general. Figure 5 (c) compares the object size (CDF) of different video types in the United States for 2010. Some video objects (*e.g.*, ASF) are very small in size and they are container objects that do not contain actual video content. Once users fetch this kind of container object, they contact media streaming servers that use non-HTTP protocols such as RTSP [55] or RTP [54] for fetching the content. The median size of such container objects is typically less than 1 KB, while that of FLV, WMV, and MPEG is 1743 KB, 265 KB, and 802 KB, respectively.

Finally, while new video streaming technologies that split a large video file into multiple smaller files for cacheability and performance started gaining popularity in late 2009 and 2010 [1, 8, 41], we do not see its wide deployment in our data set yet. With these new technologies, we expect to observe a decrease in size of video objects with an increasing number of requests. We plan to analyze this case with more recent data set in the future.

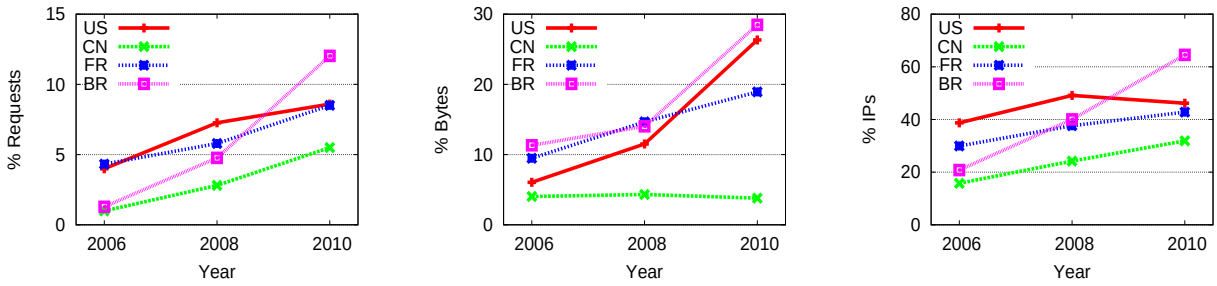
Traffic Share of Web Sites We examine the traffic share of 1) video sites [1], and 2) advertising networks and analytics sites [2] in Figure 6. We consider the top 50 sites that dominate these kinds of traffic. In Figure 6 (a), we observe that the advertising network traffic takes 1-12% of the total requests, and it consistently increases over time as the market grows [31]. In addition, we find the volume of video site traffic is consistently increasing as

[1] *e.g.*, youtube.com
[2] *e.g.*, doubleclick.com and google-analytics.com



(a) FR: JavaScript (b) CN: CSS (c) US, 2010: video

Figure 5: Object size: The object size of JavaScript and CSS becomes larger. Flash video is bigger than other video.



(a) Ads network traffic by requests (b) Video site traffic by bytes (c) Single top site % IPs

Figure 6: Top sites: Ads/video site traffic is increasing. A single top site tracks up to 65% of the user population.

shown in Figure 6 (b), taking up to 28% in Brazil for 2010. China, with lower bandwidth, sees more still image traffic than video. Finally, we see that the single top site reaches a growing fraction of all users over time in Figure 6 (c). All of the single top sites by the number of client IPs during a five-year period are either a search engine (`google.com` or `baidu.com`), or analytics site (`google-analytics.com`), reaching as high as 65% in Brazil for 2010, which may have implications for user tracking and privacy.

4. PAGE-LEVEL CHARACTERISTICS

In this section, we analyze our data with Web page-level details. We first provide background on page detection algorithms and explain the problems with previous approaches in Section 4.1. In Section 4.2, we present a new page detection algorithm called StreamStructure that is better suited for modern Web traffic analysis, and also demonstrate it is more accurate than previous approaches. Using this algorithm, Section 4.3 examines the initial page characteristics, analyzes page loading latency via simulations, and presents a simple characterization of modern Web pages.

4.1 Previous Page Detection Algorithms

A common approach for empirically modeling Web traffic is to reconstruct end-user browsing behavior from the access log data, in which users repeatedly request Web pages. Once Web pages (or main objects) are identified, we can derive relevant model parameters such as the number of embedded objects, total page size, total page time, and inter-arrival time. Thus, detecting Web page boundaries is crucial to the model accuracy.

Previous approaches for detecting page boundaries fall into two categories. The first approach (time-based) is to use the idle time between requests [9, 35, 56]. If the idle time is short enough (less than a predefined threshold), the request is assumed to be generated automatically by the browser, and it becomes an embedded object of the previous Web page. Otherwise, the request is assumed to be generated manually by the user's click, and it becomes the main object of a new Web page. The second approach (type-based) is to use the content type of the object [17]. This approach simply regards every HTML object as a main object, and any non-HTML object as an embedded object of the previous main object.

Unfortunately, the complex and dynamic nature of the current Web traffic blurs the traditional notion of Web pages, and the previous approaches do not work well. For example, client-side interactions (*e.g.*, Ajax) that usually have longer idle time would be misclassified as separate Web pages by the time-based approach. On the other hand, the type-based approach would misclassify frames in a single Web page as separate independent Web pages. As a result, these approaches would generate inaccurate traffic models if applied to modern Web traffic. Worse, they have already been used in hundreds of studies without validation.

4.2 StreamStructure Algorithm

To overcome the limitations of the previous approaches, we develop a new page detection algorithm called StreamStructure, that exploits the stream and structure information of Web pages. Our algorithm consists of three steps – grouping streams, detecting main objects, and identifying initial pages. Figure 7 depicts the definition of streams, Web pages, initial pages, main/embedded objects, and client-side interactions in our algorithm.

Step 1. Grouping Streams Instead of treating all the requests in a flat manner, we first group them into multiple independent streams by exploiting the `Referer` field. The referer of a request reveals the address of an object from which the request came from – a dependency between two objects. For example, if an HTML object includes two embedded objects in it, the referer of those two embedded objects would be the HTML object. Also, if a user clicks a link to move to a new Web page, the first request of the new Web page would have the referer field of an object in the previous Web page.

At a high level, each stream is a transitive closure of the referer relation on the set of requests. Whenever the referer of a request is empty, the request becomes the start (or root) of a new stream. If the

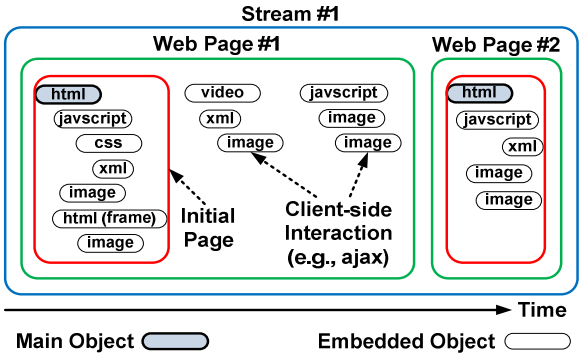


Figure 7: Streams, Web pages, initial pages, client-side interactions, and main/embedded objects.

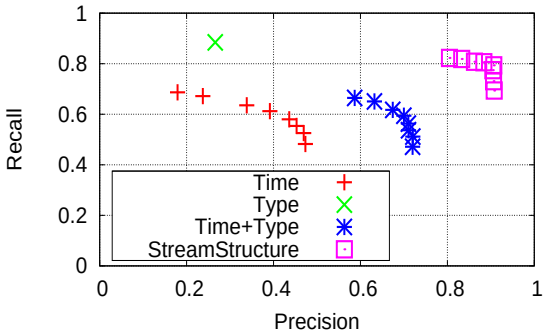


Figure 8: Precision and recall: StreamStructure outperforms previous page detection algorithms, simultaneously achieving high precision and recall. It is also robust to the idle time parameter selection.

referer of the subsequent request matches with any of the requests in the streams, we associate the request with the matched stream. In case there is more than one matched stream, we choose the latest one. If not found, we also create a new stream with the request – this happens because its referer request could be a browser cache-hit thus not present in the log.

Grouping requests with the referer relation allows isolating logs from multiple browser instances or tabs, since they belong to different streams. It also helps identifying frames and client-side interactions, since all the frames from the same Web page and all the client-side interactions to the same Web page belong to the same stream.

Even though the referer field is optional, we can safely rely on this information because most current browsers (Firefox and MSIE) enable it by default. In fact, Firefox and MSIE together account for more than 86% of our client population as discussed in Section 2. When present, we use the referer field to group requests into streams.

Step 2. Detecting Main Objects Once we finish grouping streams, we detect a main object for each stream. We first generate main object candidates by applying the type-based approach. This would find HTML frame objects as main object candidates, but non-HTML interactions would be ignored. Among those main object candidates, we discard those with no embedded object. This is based on the observation that current Web pages are typically complex, consisting of many embedded objects. We detect this by looking at the referer of the next request. If it is not the preceding main object candidate, we remove the preceding object from consideration.

Next, we apply the time-based approach to finalize the main object selection. If the idle time is less than a given threshold, it is likely that they belong to the same Web page – overlapping HTML frame objects with a short idle time would be eliminated from the selection. It is noteworthy that we consider the idle time only among the main object (HTML) candidates. This is because the interactions in a Web page happen at an arbitrary point, and it biases the idle time calculation if included. Now all the remaining objects between two main objects become the embedded objects of its preceding main object. This way, we could include all the interactions in a Web page as its embedded objects.

Step 3. Identifying Initial Pages The final task of our algorithm is to identify the initial pages, as the previous grouping still includes client-side interactions in the Web pages. The basic idea is to apply the time-based approach. However, simply checking the idle time is inaccurate because the DNS lookup [2] or browser processing time can vary significantly, especially while processing the main object before the page is fully loaded.

To this end, we exploit the popular use of Google Analytics in the pages. It is a piece of JavaScript code that collects various client-side information and reports to the analytics server when the `DOMContentLoaded` event fires. Thus, once we see this beacon in the access logs, we can safely assume that the Web page is successfully loaded at that point, and start applying the time-based approach to identify the initial page. Note that our algorithm can also use other ways than the Google Analytics beacon to detect the page loading event. For example, one could utilize beacons from other analytics services, or even instrument Web pages with custom JavaScript at the proxy.

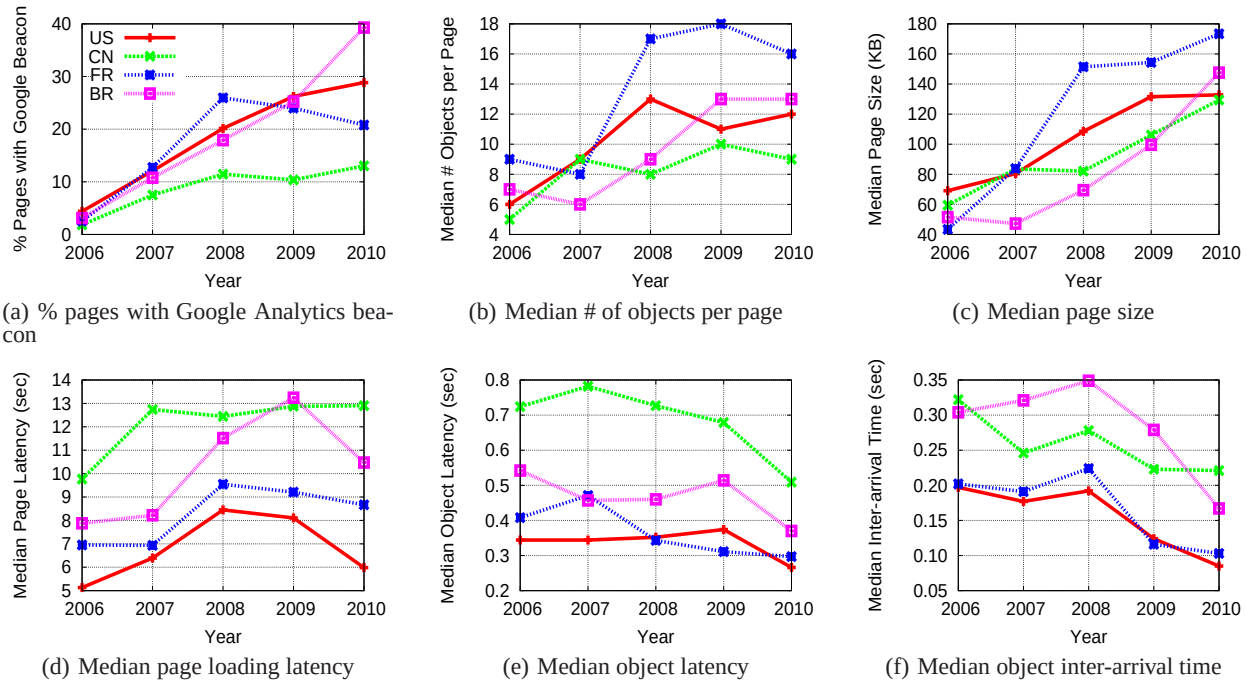
Validation We validate the accuracy of StreamStructure and the existing approaches on the manually collected data set by visiting (via CoDeeN) the top 100 sites of Alexa's list [5] with MSIE. We not only visit the top-level pages, but also follow approximately ten links from those top-level pages for each site, resulting 1,197 Web pages in total. [3] We also record the URLs of those visited Web pages (or main objects), and compare them with the URLs of the Web pages found by each approach. Note that this data collection is different from actual browsing patterns, but we believe that it is sufficient to capture the structure of representative Web pages and thus useful for validation.

Figure 8 shows the precision and recall of various approaches. Precision is defined as the number of correct Web pages found divided by the total number of Web pages found, and recall is defined as the number of correct Web pages found divided by the total number of correct Web pages. For comparison, we also try a simple combination of time-based and type-based approaches (Time+Type) that does not exploit the stream and structure information. Multiple data points represent the results of various idle time (0.1, 0.2, 0.5, and 1–5 seconds) parameters.

The time-based approach performs in general very poorly, and the best result achieves only a precision of 0.45 and a recall of 0.55. Also, the performance is very sensitive to the idle time selection. The type-based approach shows the highest recall above 0.88, which implies that the main objects of most Web pages are HTML. However, the precision is very low, only about 0.27. StreamStructure outperforms all of the previous approaches, achieving high precision and recall above 0.8 simultaneously. Furthermore, it is quite robust to the idle time parameter selection. The time+type approach is less accurate, proving the importance of exploiting the stream and structure information.

Finally, we investigate the sensitivity of the idle time parameter

[3] Some sites have many Web pages while others do not.



(a) % pages with Google Analytics beacon (b) Median # of objects per page (c) Median page size

(d) Median page loading latency (e) Median object latency (f) Median object inter-arrival time

Figure 9: Initial page characteristics: The Google Analytics beacon gets increasingly popular, and pages get bigger both in terms of the size and number of objects. On the other hand, the page loading latency has dropped in 2009 and 2010 because of the increased number of concurrent connections and reduced object latency.

for identifying initial pages by comparing CDFs of the page loading time, number of objects, and size of the initial pages with different idle time thresholds. Overall, 23.9% of pages have Google Analytics beacons in our manually collected Alexa data set. We observe that an idle time of 0.1 seconds is too short and 5 seconds is too long, distorting the distribution significantly. On the other hand, an idle time between 0.5 and 2 seconds generates quite stable and similar results.

4.3 Analysis Results

We apply the StreamStructure algorithm to our CoDeeN access log data set, and analyze the derived Web pages in various aspects. We choose the idle time of one second both for identifying Web pages out of streams, and for identifying initial pages out of Web pages. Among all the users, we ignore those who are active for less than 30 minutes to reduce potential bias. We first examine the characteristics of initial pages, and analyze the page loading latency in detail via simulations. Finally, we provide a simple characterization of modern Web pages including client-side interactions.

Initial Page Characteristics We first show the fraction of Web pages that have a Google Analytics beacon in our data set in Figure 9 (a). It is less than 5% in 2006, but it has become increasingly popular and accounts for about 40% in 2010. While there is a little variation over time, the volume of the initial page traffic roughly accounts for about 40-60% of the entire Web traffic in terms of both requests and bytes. The rest of the traffic is client-side interactions, which is quite a significant amount.

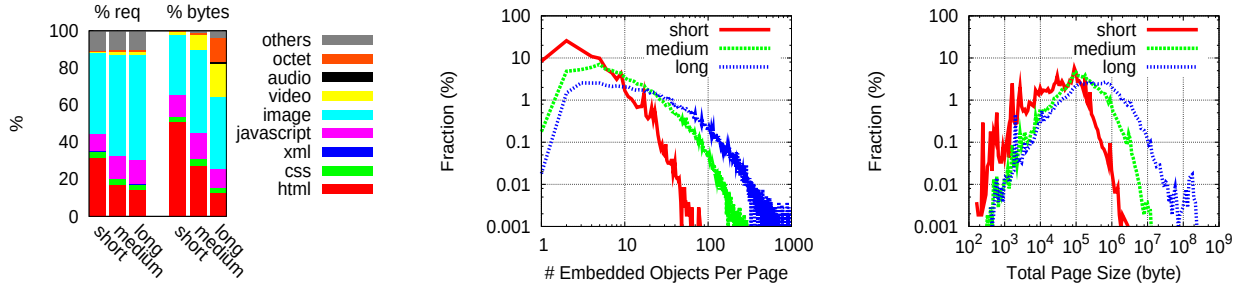
In Figure 9 (b)-(f), we examine the changes in the number of objects, size, and latency of the initial pages, and the latency and inter-arrival time of each individual object in the initial pages. We compare the median values rather than the mean, since our page detection algorithm is not perfect and the mean is more susceptible to outliers/errors.

First of all, the pages have become increasingly complex where we observe a consistent increase of the number of objects and the total size in Figure 9 (b) and (c). For example, the median number of objects per page in the United States sees an increase from 6 objects in 2006 to 12 objects in 2010, and the median page size gets bigger from 69 KB in 2006 to 133 KB in 2010. This increase is likely related to the popular use of advertisement/analytics, and the object size increase of JavaScript and CSS in Figure 5 (a) and (b).

While the page loading latency also sees a general increase in Figure 9 (d) until 2008, instead we see it decrease in 2009 and 2010. For example in the United States, the median latency increases from 5.13 seconds in 2006 to 8.45 seconds in 2008, but it decreases to 5.98 seconds in 2010. [4] This decrease likely stems from the increased number of concurrent connections in Figure 3. Another decreasing factor is the reduced latency of fetching an object in Figure 9 (e), and it also makes the object inter-arrival rate burstier in Figure 9 (f). As the object size does not get smaller over time, the decrease of the object latency is likely related to the improved client bandwidth in Figure 1, as well as the improved caching behavior of many Web sites.

Page Loading Latency Simulation As the page loading latency is determined by many factors including the number of concurrent connections per server, object latency, and dependency among objects, we examine the impact of these factors via simulations. Each object is fetched from a central FIFO queue, and we use the measured object latency in the access logs for the simulated object latency. The object dependency is extracted from the referer relations. Since the purpose of this simulation is to assess which factors affect the page loading latency rather than to predict the actual latency, we simplify the simulation as follows. First, we ignore network latency and browser parsing/processing time – there is no net-

[4]The actual user-perceived latency is smaller than our measured latency because users recognize pages to be loaded before all of the embedded objects are completely downloaded.



(a) Page content type composition (b) # of objects (c) Total size

Figure 11: Characterization of entire Web pages from the United States in 2010: We define short (0-25th percentile), medium (25-75th), and long (75-100th) pages by total time. Short pages are bursty and HTML-oriented while long pages are video/binary-oriented and involve heavy client side interactions.

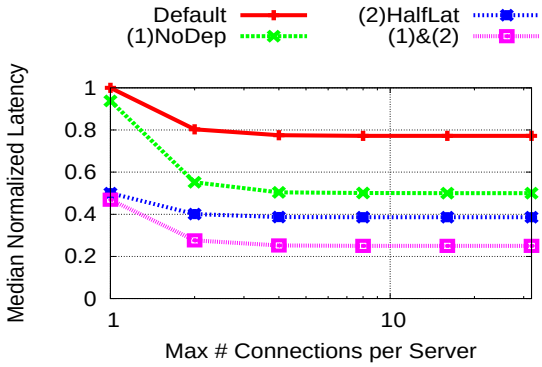


Figure 10: Page loading latency simulation (US, 2010): Increasing the number of simultaneous connections could further reduce page load latency by 23%. Removing dependencies between objects (`NoDep`) would yield at most a 50% reduction. Reducing per-object latency by 50% (`HalfLat`) would actually reduce page-load latency by 67% because of the simultaneous connections. Together, page loading latency can be reduced by up to 75%.

work idle time during the page loading process. Second, any dependent object is not fetched before its parent object is finished, which is less aggressive than current practice, where a browser typically starts fetching embedded objects as soon as it finds their URLs.

Figure 10 presents the median simulated latency from the United States in 2010, as a function of the maximum number of concurrent connections per server. We omit the results of other countries that are very similar to that of the United States. We simulate four different scenarios, and the latency is normalized by the default latency with one concurrent connection per server. First, we observe that increasing the number of concurrent connections per server would reduce the latency by up to 23% at 8 connections, beyond which we see no benefit. Second, we simulate the ideal case where there is no object dependency (`NoDep`) – all of the object URLs are known in advance, and it reduces the latency by up to 50%. Given this latency reduction, it is worth exploring ways to relieve/eliminate the object dependency. Third, if per-object latency is reduced by half (`HalfLat`) via better caching/prefetching approaches, it could actually reduce the page loading latency by up to 61% due to the simultaneous connections. All together, we observe that page loading latency could be reduced by up to 75%.

Entire Page Characteristics For a simple characterization of modern Web pages, we divide all of the Web pages including client-side interactions into three groups based on the total page time – short (0-25th percentile), medium (25-75th), and long (75-100th) pages. We then characterize these pages in terms of the number of embedded objects, page size, object inter-arrival time, and content type distribution. Figure 11 presents an example characterization of Web pages from the United States in 2010.

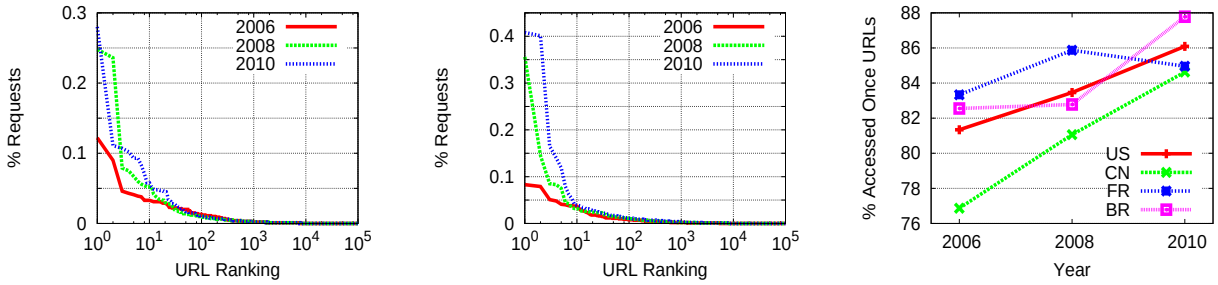
Overall, long pages consume about 55% of total requests, and medium pages take about 40%. In terms of bytes, long pages take even more than 60%. Short pages account for only about 5% in terms of both requests and bytes. The content type distribution in Figure 11 (a) reveals the characteristics of the pages more clearly. Short pages are mainly HTML-oriented, and search activities could be typical examples. On the other hand, long pages show a higher percentage of video and octet-stream bytes than others, meaning these are mainly video watching activities and large file downloads. Medium pages lie in between, and typical examples would be browsing news or blogs.

In terms of the number of embedded objects, short pages mostly have less than 10 objects, and medium and long pages have a larger number of embedded objects, as in Figure 11 (b) where we show PDFs. The median is 4, 12, and 30 for short, medium, and long pages, respectively. Especially, we observe heavy client-side interactions in long pages. Note that medium pages will often specify dozens of embedded images in their HTML but as Web sites improve their cacheability best practices, most of these are cached at the browser, and we observe only a median of 12 fetches for the updated portions of their content. This in part also explains why page loading latency is improving despite the increase in page complexity.

In addition, Figure 11 (c) shows PDFs of the total page sizes, and we observe that the median difference is about 3x between short (40 KB) and medium pages (122 KB), and more than 2x between medium and long pages (286 KB). Note that long pages have a very long tail reaching up to 370 MB, while the largest page size is only about 5 MB for short pages and 13 MB for medium pages. Finally, we observe that short and medium pages are burstier than long pages as it does not usually involve client-side interactions. The median object inter-arrival time is 90, 89, and 114 ms for short, medium, and long pages, respectively.

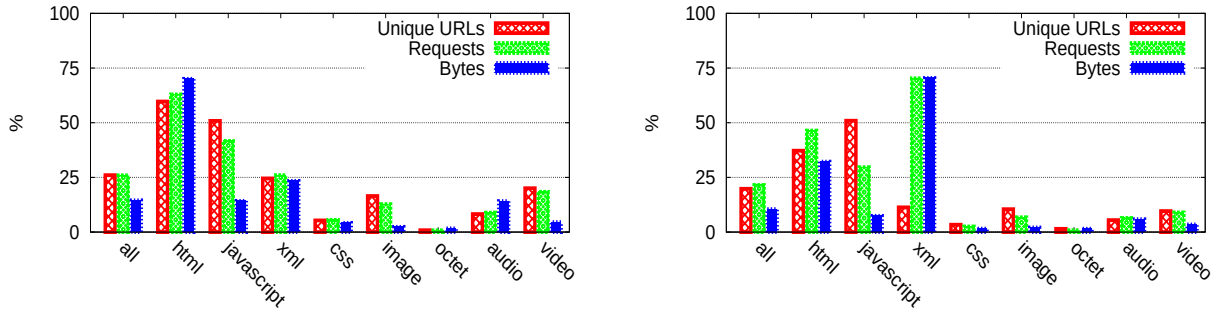
5. REDUNDANCY AND CACHING

The last part of our study is to analyze the redundancy in Web traffic and the impact of caching. For this study, we analyze the full content of traffic as well as the access logs. Table 2 shows the summary of our content data set from April, 2010. We capture cache misses only to account for simple improvements like using a local proxy cache. Throughout this section, our analysis is based



(a) US: Top 100K URLs by % requests (b) CN: Top 100K URLs by % requests (c) % accessed once URLs (tail)

Figure 12: URL popularity: The popular URLs grow, but the long tail of the content is also growing.



(a) US (b) CN

Figure 13: Uncacheable objects by content types: Overall, 19.8-32.2% of unique URLs, 21.5-28.3% of total requests, and 10.5-14.8% of total bytes are uncacheable. HTML and JavaScript are dynamically generated and thus less cacheable.

	USA	China	France	Brazil
# Requests (K)	8,611	12,036	2,129	4,018
Volume (GB)	198	218	42	79

Table 2: Summary of captured full content data (cache-misses only)

on logically centralized but physically distributed proxy systems, just like CoDeeN.

We first provide the details of content-based caching in Section 5.1. Using the access log data, we examine the changes in URL popularity during the five-year period in Section 5.2. Using the full content data, we directly compare the effectiveness of object-based caching and content-based caching in Section 5.3, and quantify the origins of redundancy in Section 5.4. We also calculate the actual byte hit rates with practical cache storage sizes in Section 5.5. Finally, we analyze the characteristics of aborted transfers, and discuss its caching implications in Section 5.6.

5.1 Content-based Caching

At a high level, content-based caching works by splitting an object or file into many smaller chunks, and caching those chunks instead of an entire object. The chunk boundaries are determined based on the content, commonly with Rabin fingerprinting [48] – if the fingerprinting value over a sliding window of data matches with low order n bits of a predefined constant K, this region of data constitutes a chunk boundary. The expected average chunk size is 2^n byte assuming a uniform distribution of content values. To prevent chunks from being too small or large in a pathological case, we specify the minimum and maximum size of chunks as well. Unlike fixed-size chunking (e.g., every 1 KB), content-based chunking is robust to any insertion/deletion/modification to the content since it only affects nearby chunks.

Once chunk boundaries are detected, chunks are named based on the content, often with SHA-1 hash. The next time the system sees the same chunk, it can pass only a reference instead of the original content. This way, content-based caching could find the same content within an object and across different objects, yielding much higher cache hit rates than object-based caching. Furthermore, it is protocol independent and effective for uncacheable content as well.

5.2 URL Popularity

We investigate the underlying changes in URL popularity during the five-year period with our access log data set, which directly influences the caching effectiveness. We find two interesting trends. First, we observe that the popular URLs are getting more popular as in Figure 12 (a) and (b) where we present the request percentage of the top 100,000 URLs in the United States and China. The request traffic to most popular URL increases from 0.08-0.12% in 2006 to 0.28-0.41% in 2010, and this concentration would increase the cache hit rate. The most popular URL in the United States for 2010 is a dynamically generated beacon object from `google.com`, which is uncacheable, though. At the same time, we also find that the percentage of URLs that are accessed only once is consistently increasing as in Figure 12 (c). We see its increase from 76.9-83.3% in 2006 to 84.6-87.8% in 2010. Overall, they account for a significant amount of traffic – 30.0-48.8% of total requests and 27.3-63.9% of total bytes. These least popular URLs are all cache-misses and would decrease the cache hit rate.

While these two trends in URL popularity could affect cache hit rate both positively and negatively, we do not observe any consistent changes in resulting cache hit rate during the five-year period. This is because they cancel out each other, and cache hit rate is also determined by other factors such as user population. In order to get an upper bound on object-based cache hit rate with our access log data set, we assume every object is cacheable, and two objects are identical (cache hit) once their URLs and content lengths

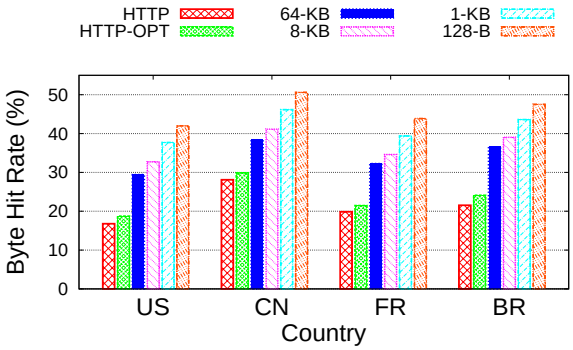


Figure 14: Ideal cache hit rate with infinite cache storage: Content-based caching with 128-bytes chunks achieves almost 2x larger byte hit rate than object-based HTTP caching.

match. The estimated cache hit rate we observe ranges from 35.6% to 54.5%, and the byte hit rate ranges from 15.1% to 49.3%. The byte hit rate is smaller than the cache hit rate because cache hits are biased towards relatively smaller objects. In fact, we observe that the mean object size of those URLs that are accessed only once is always larger than the mean object size of those URLs that are accessed more than once over the five years.

5.3 Caching Effectiveness

In this section, we first investigate HTTP cacheability of the traffic, and calculate the ideal cache hit rate. We then compare the effectiveness of object-based and content-based caching on our full content data, and further examine the impact of content types.

HTTP Cacheability We examine HTTP cacheability of objects with our full content data from 2010. We decide if an object is cacheable or not by looking at its `Cache-Control` and `Pragma` fields in the response header. Figure 13 shows the percentage of uncacheable objects in terms of the number of unique URLs, total requests, and total bytes. Overall, 19.8-32.2% of unique URLs are uncacheable, and it accounts for 21.5-28.3% of total requests and 10.5-14.8% of total bytes. Among different content types, HTML and JavaScript are less cacheable than other content types, implying that they are dynamically generated and updated frequently. Even though the low fraction of uncacheable traffic implies substantial potential for caching, the actual cache hit rates would be severely limited due to the growing fraction of URLs that are accessed only once.

We also observe a few other interesting points. First, a significant portion of XML traffic (over 70%) in China is uncacheable, and it turns out to be due to the popular use of Really Simple Syndication (RSS) [52] feeds – two RSS URLs are responsible for 90.8% of total uncacheable bytes and 64.8% of total uncacheable requests. Second, Brazil (not shown) shows a higher fraction of uncacheable XML and audio traffic than other countries. This is due to the popular use of real time update of sports games and live streaming of audio.

Ideal Byte Hit Rate We calculate the ideal bandwidth savings achievable with a centralized proxy cache having infinite cache storage by the traditional object-level HTTP caching and content-based caching. For object-level caching, we decide if an object is cacheable by respecting cache-control headers. If cacheable, we check if the URLs and content lengths match as in Section 5.2. We also calculate a slightly optimistic behavior of object-based caching by discarding query strings from URLs in order to accommodate the case where two URLs with different metadata actually belong to the same object. For content-based caching, we vary the average chunk size from 128 bytes, 1 KB, 8 KB, to 64 KB. Note that we apply content-based caching on compressed content without decompressing it, because the volume of compressed content such as `gzip` or `deflate` is less than 1% in our data set.

In Figure 14, we observe that content-based caching with any chunk size outperforms object-based caching. The cache hit rate of object-level caching ranges from 27.0-37.1% (not shown in the figure), but the actual byte hit rate is only 16.8-28.1%, which is lower than the byte hit rates from a decade ago, but similar to that in more recent studies [3, 13, 24, 30, 40, 63]. The hit rate of the optimistic version (`HTTP-OPT`) is only slightly larger. On the other hand, the lowest byte hit rate of content-based caching is 29.4-38.4% with 64 KB chunks, and the highest byte hit rate is 42.0-50.6% with 128 byte chunks, 1.8-2.5x larger than object-level caching's byte hit rate. The small chunk size performs better than the large chunk sizes because of its finer granularity. For example, 128 bytes chunks can detect redundancy at the sentence-level, but 64 KB can do only at the document-level.

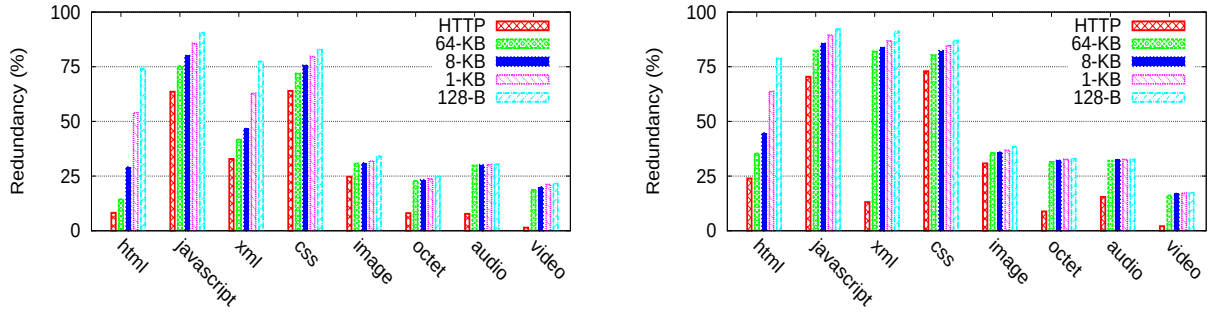
Impact of Content Types Among many different content types, we find that text resources such as HTML, JavaScript, XML, and CSS have much higher redundancy than binary resources such as image, audio, video, and octet-stream. Figure 15 shows the ideal redundancy by content type. In particular, JavaScript shows over 90% of redundancy with the smallest chunk size of 128 bytes. On the other hand, video exhibits much lower redundancy of 20%, illustrating the impact of long-tailed popularity in video content. Object-based caching performs very poorly, and its redundancy elimination for XML is one-eighth that of the gains with 128 byte chunks in China.

We also find that content-based caching works well regardless of the content types, while the object-based caching is mainly effective for JavaScript and image traffic only. Figure 16 depicts the contribution of byte savings, basically showing which caching scheme works best for which content type. In object-based caching, the contribution of JavaScript and image is relatively larger than that of other content types. It should be noted that the contribution of binary resources such as video, audio, and octet-stream is extremely low, implying that object-based caching is not suitable for them. On the other hand, content-based caching provides more uniform benefits.

5.4 Origins of Redundancy

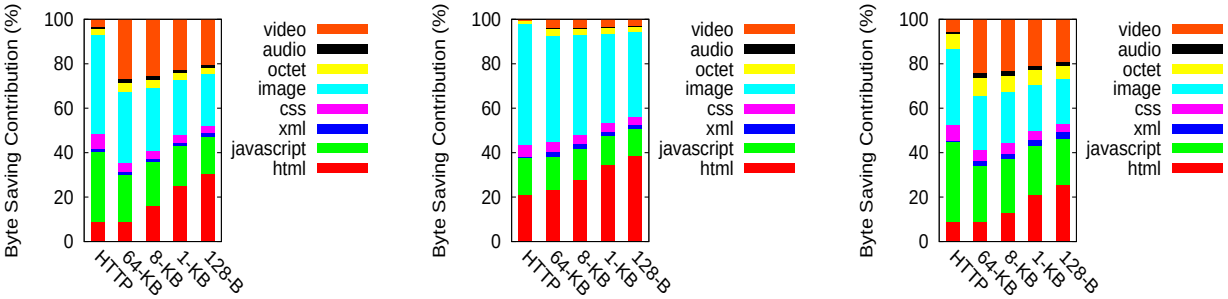
In order to understand how content-based caching approaches provide more than double the byte hit rate than the object-based caching approach, we quantify the contribution of redundancy from different sources in Figure 17. We use the average chunk size of 128-bytes for this analysis.

Overall, we observe that about 40.3-58.6% of the total redundancy is due to identical objects with the same URLs, which is essentially the upper bound of the object-based caching approach's byte hit rate (`object-hit`). The other half of the total redundancy is purely from the content-based caching approaches, and we further break it into the following three sources. First, there exists redundancy across the content changes of an object (`intra-URL`), and it accounts for about 21.8-32.5% of the total redundancy. Second, some objects with different URLs actually have identical content [32] (`aliasing`), and it represents 6.7-9.8% of the total redundancy. Finally, the rest is due to the redundancy across different objects that have different URLs and non-identical content (`inter-URL`), and it represents 12.8-20.0% of the total redundancy. This analysis result implies that most of the additional savings from the content-based caching approaches come from its abil-



(a) US (b) CN

Figure 15: Ideal redundancy by content types: Text resources have higher redundancy than binary.



(a) US (b) CN (c) BR

Figure 16: Byte saving contribution by content types: Content-based caching is effective for any content type, but object-based caching works well only for JavaScript and image.

ity to detect and eliminate the redundancy in the content changes of an object as well as redundancy across different objects.

In terms of content types, we find that HTML and XML generally show relatively higher intra-URL redundancy than other content types. It implies that they are frequently updated but their content changes slowly. Also, aliasing in general accounts for a small amount of the total redundancy, but we observe a significant amount of aliasing in XML and audio content types in Brazil. This is again because of the popular use of the real time updates of sports games (XML) and live streaming of audio in Brazil. These objects have identical content but with different URLs. Finally, we see that most of the redundancy in binary resources, especially video, come from partial content overlaps (intra-URL + inter-URL) rather than complete object matches (object-hit + aliasing). This is partly because they are aborted before they are fully downloaded. We examine the aborted transfers in more detail in Section 5.6.

5.5 Cache Storage Size

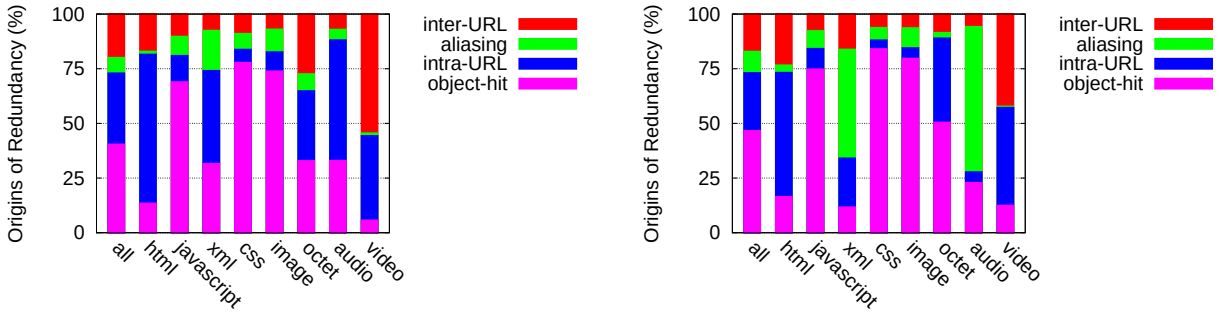
We simulate cache behavior with different cache storage sizes to determine the required cache storage size for achieving close to the ideal byte hit rate, but also include the metadata overhead (20 bytes per chunk) of content-based caching in the byte hit rate calculation. We use a simple LRU cache replacement policy as a first step, and leave for future work investigating more sophisticated policies [46].

In addition to object-based and content-based caching, we also simulate multi-resolution chunking (MRC), a recently-developed strategy that simultaneously exploits multiple chunk sizes [28] and is well-suited for large storage sizes. MRC always favors large chunks over small ones, and uses small chunks only when large chunks are cache misses. It also caches all different chunk sizes for the same content redundantly for the future reference. This way, MRC minimizes the metadata overhead, disk accesses, and memory pressure at the cost of more disk space.

Figure 18 shows our simulation results in the United States and China, which shows that content-based caching always outperforms object-based caching regardless of cache storage size. However, due to the significant metadata overhead for fixed 128 bytes chunks, the actual byte hit rate of 128 byte chunks is similar to that of 1 KB chunks. The saturation point of cache size is similar across the different caching approaches except for MRC. For example, beyond 100 GB of cache storage, the byte hit rate no longer increases in the United States and China. The saturation point essentially indicates the working set size of the traffic, so increasing the cache size beyond it gives no further benefits. On the other hand, while MRC performs relatively poorly when cache storage is small, it continues to increase the byte hit rate beyond the saturation point, as the multiple chunk sizes reduce metadata overhead. The simulation has a few missing data points because of the limitation of main memory we have (16 GB) during the simulation. Also, the byte hit rate of MRC with infinite cache size is estimated from the ideal byte hit rate of 128 byte chunks minus 1% overhead.

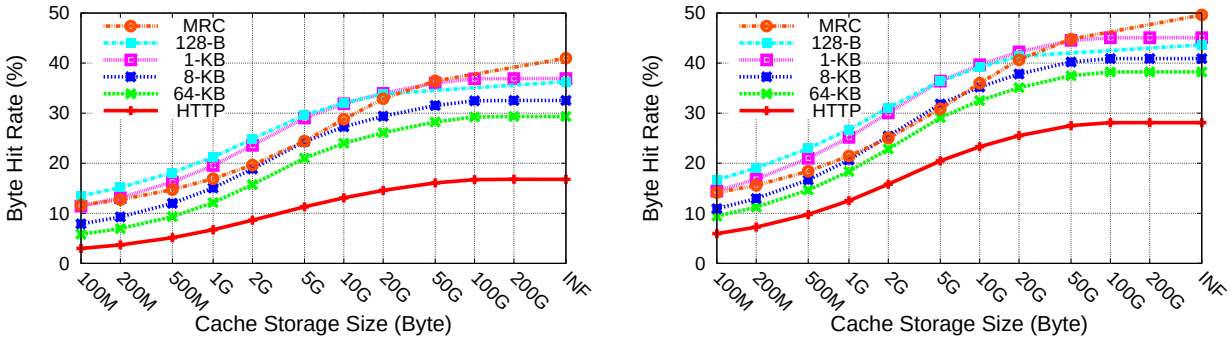
While increasing cache storage size gives diminishing returns for object-based caching, using large cache storage with MRC is highly beneficial as it doubles the byte hit rate compared to object-based caching. This option would be especially attractive in developing regions where the bandwidth is much more expensive than disk storage [27]. Since a TB-sized disk costs less than $100, it makes sense to allocate much more cache storage than was used 10 years ago, when disk sizes were in the tens of GB.

In our data set, we need about 800 GB for the United States and China, 200 GB for France, and 400 GB for Brazil to achieve close to the ideal byte hit rate with MRC. It is roughly four times of the total traffic size because MRC uses four different chunk sizes in



(a) US (b) BR

Figure 17: Origins of redundancy with 128-bytes chunks: Most of the additional savings from the content-based caching approaches come from partial content overlap – the redundancy across different versions of an object as well as redundancy across different objects.



(a) US (b) CN

Figure 18: Cache size vs. byte hit rate: A large cache with MRC provides 2x the byte hit rate than the object-based caching.

	Request (K)	Byte (GB)	GB if fully downloaded
US	265 (3.1%)	61 (30.8%)	712 (83.8%)
CN	377 (3.1%)	27 (12.4%)	444 (69.9%)
FR	38 (1.8%)	10 (23.6%)	258 (88.8%)
BR	85 (2.1%)	22 (28.3%)	216 (79.3%)

Table 3: Aborted transfers

our simulation. Note that one might want to reduce the storage requirement by storing only unique content and metadata such as offset for different chunk sizes. However, it complicates the cache index management and increases memory pressure, as pointed out by Ihm *et al.* [28].

5.6 Aborted Transfers

In Table 3, we find a small number of requests (1.8-3.1%) are aborted before they are fully downloaded, but their volume is quite significant. These events occur when users cancel ongoing transfers by clicking the stop button of the browser, or move to another Web page. We detect the aborted transfer if the downloaded length is less than the given content-length in the response header. The total volume of the downloaded bytes until aborted is 12.4-30.8%. If they were fully downloaded, it would take 69.9-88.8% of the entire traffic. The investigation of the content type distribution of these transfers reveals that most of the bytes are from the video transfers, presumably previewing the first part of the video clips. In particular, Flash video comprises roughly 40-70% of all aborted transfers, and we also observe users canceling file downloads.

The large volume of aborted transfers could negatively impact the performance of object-based caching proxies. Such systems have roughly four options to deal with the aborted transfers. The first option is to discard and do not cache them, but it just wastes the bandwidth and reduces cache hit rate. The second option is to fully download and cache them (of course, for those cacheable objects only), but it consumes significant bandwidth for downloading objects that might not be referenced in the future. The third option lies in between the first and second, and decides whether to discard or fully download depending on the number of bytes remaining [58]. The final option is to cache the downloaded portion and do a range request on a cache hit, but it is effective only for cacheable objects.

In comparison, content-based caching could cache data from only the downloaded content without any configuration, thus any data received over network, even uncacheable, is useful. As evidence, content-based caching's byte hit rate of video traffic is much higher than object- based caching's byte hit rate in Figure 15.

6. RELATED WORK

Our work is related to previous work in the areas of Internet monitoring, traffic characterization, and caching behavior. We describe our relation to this previous work below.

Internet Monitoring There is a great deal of previous work analyzing traffic behavior. For example, Akamai analyzes data gathered from its global server network, and reports the Internet penetration rate and connection speeds for each country [10]. Also, ipoque examines traffic from its customer ISPs and universities [29], and they find the increase of Web traffic as well as the decrease of P2P traffic due to the popularity of file hosting, social networking, and video streaming sites. Several other studies commonly observe the same trend of increasing Web and video traffic [21, 34, 36].

While all of these previous studies primarily focus on the anal-

ysis of overall usage of Internet traffic, our focus is to investigate various aspects of Web traffic changes in great detail. A few other studies also have conducted long-term characterizations of Web traffic [14, 26], but their analyses on the data set from specific organizations, such as universities or research institutes, are tied to their host institutes. Instead, our large-scale data set spanning a multi-year period covers a world-wide user population.

Web Traffic Characterization A very widely used Web traffic model was first proposed by Mah, in which he introduces the idle time based page detection algorithm [35]. Since then, this model has been widely adopted by many researchers for characterizing Web traffic [9, 56]. Later, Choi and Limb developed a method that simply regards every HTML object as a Web page [17]. More recently, several studies have investigated a small number of popular Ajax Web applications such as maps and Web mails, and streaming services [16, 53].

However, all of the previous studies have limitations in that they either assume the simple/static Web pages ignoring client-side interactions, or rely on application/site-specific knowledge. Instead, our page detection algorithm is able to identify initial pages and client-side interactions, and also does not require application/site-specific knowledge. Furthermore, we demonstrate that our algorithm is more accurate than the previous approaches via careful validation.

A contemporary work by Butkiewicz *et al.* [12] investigates the complexity of Web sites with browser-based active measurements from four vantage points. While their use of HTTP archive record (HAR) format [25] allows a precise detection of page load events, their data set consists of only the top-level pages of randomly chosen 2,000 Web sites, also ignoring client-side interactions. Analyzing real users' browsing behaviors with detailed HAR logs would be an interesting future work.

Redundancy and Caching Traditional object-level Web caching works by storing previously seen objects and serving them locally for future requests. However, the benefit of object-based caching is limited only to the cacheable objects such as static text and image files – the typical cache hit rates reported in the previous work range from 35% to 50% in much earlier work, and have dropped over time [3, 13, 24, 30, 40, 63]. The byte hit rate is even worse as a cache hit is biased towards smaller popular objects. Most recently, Ager *et al.* examined potential for HTTP caching in various scenarios by controlling the strictness of object cacheability [3]. More advanced object-based caching techniques include delta-encoding that reduces traffic for object updates [39], and duplicate transfer detection (DTD) that avoids downloading of aliased objects [40].

Spring and Wetherall further extend object-based caching to sub-packet granularity, and develop a protocol independent content-based caching technique [57]. Since then, it has been adapted in many applications – network file systems [7, 42], WAN acceleration [28, 50], Web caching [43, 49], and storage systems [20]. Recently, Anand *et al.* analyzed university and enterprise network traces, and show that 15-60% of the entire traffic is redundant, while the redundancy of Web traffic is only 16-32% [6].

While both the object-based and content-based caching schemes have been studied heavily, the focus of our work is to perform a head-to-head comparison between them on real Web traffic. Our analysis result shows that content-based caching achieves byte hit rates of 42-51%, almost twice that of object-based caching's byte hit rates. Furthermore, we evaluate the effectiveness of MRC [28], and find increasing cache storage size is highly beneficial. Indeed, the redundancy we find (42-51%) is much higher than what Anand *et al.* report (16-32%), and it is partly because we assume a large disk-based cache while they use in-memory cache only.

7. CONCLUSIONS

For a better understanding of modern Web traffic, we analyze five years of real Web traffic from a globally distributed proxy system that captures the browsing behavior of over 70,000 daily users from 187 countries. Among our major findings is that Flash video and Ajax traffic is consistently increasing, and search engine/analytics sites are tracking an increasingly large fraction of users. Our StreamStructure algorithm reveals that almost half the traffic now occurs not as a result of initial page loads, but as a result of client-side interactions after the initial load. Also, while pages have grown in terms of both the number of objects and size, page loading latency has dropped due to the increased number of concurrent connections and improved caching behavior. Finally, multi-resolution chunking (MRC) with large cache storage provides almost twice the byte hit rate of traditional object-based caching, and it is also effective for aborted transfers. Most of the additional savings of content-based caching are due to the partial content overlaps.

8. ACKNOWLEDGMENTS

We would like to thank our shepherd, Steve Uhlig, as well as the anonymous IMC reviewers. We are grateful to KyoungSoo Park for keeping CoDeeN running all these years and for providing us with the logs. We also thank Eric Keller, Wonho Kim, and Siddhartha Sen for their helpful comments on an earlier version of this paper. This research was partially supported by the NSF Awards CNS-0615237 and CNS-0916204.

References

[1] Adobe HTTP Dynamic Streaming. http://www.adobe.com/products/httpdynamicstreaming/.

[2] B. Ager, W. Mühlbauer, G. Smaragdakis, and S. Uhlig. Comparing DNS resolvers in the wild. In *Proc. Internet Measurement Conference*, Melbourne, Australia, Nov. 2010.

[3] B. Ager, F. Schneider, J. Kim, and A. Feldmann. Revisiting cacheability in times of user generated content. In *Proc. 13th IEEE Global Internet Symposium*, San Diego, CA, Mar. 2010.

[4] AJAX - Connectivity Enhancements in Internet Explorer 8. http://msdn.microsoft.com/en-us/library/cc304129(v=vs.85).aspx.

[5] Alexa the Web Information Company. http://www.alexa.com/.

[6] A. Anand, C. Muthukrishnan, A. Akella, and R. Ramjee. Redundancy in network traffic: Findings and implications. In *Proc. ACM SIGMETRICS*, Seattle, WA, June 2009.

[7] S. Annapureddy, M. J. Freedman, and D. Mazières. Shark: Scaling file servers via cooperative caching. In *Proc. 2nd USENIX NSDI*, Boston, MA, May 2005.

[8] Apple HTTP Live Streaming. http://developer.apple.com/resources/http-streaming/.

[9] P. Barford and M. Crovella. Generating Representative Web Workloads for Network and Server Performance Evaluation. In *Proc. ACM SIGMETRICS*, Madison, WI, June 1998.

[10] D. Belson. Akamai state of the Internet report, q4 2009. *SIGOPS Oper. Syst. Rev.*, 44(3):27–37, 2010.

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

[12] M. Butkiewicz, H. V. Madhyastha, and V. Sekar. Understanding website complexity: Measurements, metrics, and implications. In *Proc. Internet Measurement Conference*, Berlin, Germany, Nov. 2011.

[13] R. Cáceres, F. Douglis, A. Feldmann, G. Glass, and M. Rabinovich. Web proxy caching: the devil is in the details. In *Proc. 1st ACM Workshop on Internet Server Performance*, Madison, WI, June 1998.

[14] T. Callahan, M. Allman, and V. Paxson. A longitudinal view of HTTP traffic. In *Passive & Active Measurement (PAM)*, Zurich, Switzerland, Apr. 2010.

[15] M. Casado and M. J. Freedman. Peering through the shroud: the effect of edge opacity on IP-based client identification. In *Proc. 4th USENIX NSDI*, Cambridge, MA, Apr. 2007.

[16] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon. I Tube, You Tube, Everybody Tubes: Analyzing the World's Largest User Generated Content Video System. In *Proc. ACM SIGCOMM Internet Measurement Conference*, San Diego, CA, USA, Oct. 2007.

[17] H.-K. Choi and J. O. Limb. A behavioral model of Web traffic. In *IEEE International Conference on Network Protocols (ICNP)*, Toronto, Canada, Oct. 1999.

[18] D. Crane, E. Pascarello, and D. James. *Ajax in Action*. Manning Publications Co., Greenwich, CT, USA, 2005.

[19] Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. http://www.w3.org/TR/CSS2/.

[20] Data Domain. http://www.datadomain.com/.

[21] J. Erman, A. Gerber, M. T. Hajiaghayi, D. Pei, and O. Spatscheck. Network-aware forward caching. In *Proc. 18th International World Wide Web Conference*, Madrid, Spain, May 2009.

[22] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. *Hyptertext Transfer Protocol – HTTP/1.1*. Internet Engineering Task Force, June 1999. RFC 2616.

[23] Top 7 Firefox Proxy Addons. http://www.techiezine.com/top-7-firefox-proxy-addons/.

[24] S. Gribble and E. Brewer. System Design Issues for Internet Middleware Services: Deductions from a Large Client Trace. In *Proc. 1st USENIX Symposium on Internet Technologies and Systems (USITS)*, Monterey, CA, Dec. 1997.

[25] HTTP Archive Specification. http://groups.google.com/group/http-archive-specification/web/har-1-2-spec.

[26] F. Hernandez-Campos, K. Jeffay, and F. Smith. Tracking the evolution of Web traffic: 1995-2003. In *Proc. IEEE/ACM MASCOTS*, Oct. 2003.

[27] S. Ihm, K. Park, and V. S. Pai. Towards Understanding Developing World Traffic. In *Proc. 4th ACM Workshop on Networked Systems for Developing Regions (NSDR)*, San Francisco, CA, June 2010.

[28] S. Ihm, K. Park, and V. S. Pai. Wide-area Network Acceleration for the Developing World. In *Proc. USENIX Annual Technical Conference*, Boston, MA, June 2010.

[29] ipoque. Internet Study 2008/2009. http://www.ipoque.com/resources/internet-studies/internet-study-2008_2009.

[30] D. L. Johnson, E. M. Belding, K. Almeroth, and G. van Stam. Internet usage and performance analysis of a rural wireless network in Macha, Zambia. In *Proc. 4th ACM Workshop on Networked Systems for Developing Regions (NSDR)*, San Francisco, CA, June 2010.

[31] JPMorgan Chase & Company. The Rise of Ad Networks. http://www.mediamath.com/docs/JPMorgan.pdf.

[32] T. Kelly and J. Mogul. Aliasing on the World Wide Web: prevalence and performance implications. In *Proc. Eleventh International World Wide Web Conference*, Honolulu, Hawaii, USA, May 2002.

[33] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson. Netalyzr: illuminating the edge network. In *Proc. Internet Measurement Conference*, Melbourne, Australia, Nov. 2010.

[34] C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, and F. Jahanian. Internet inter-domain traffic. In *Proc. ACM SIGCOMM*, New Delhi, India, Aug. 2010.

[35] B. A. Mah. An Empirical Model of HTTP Network Traffic. In *Proc. IEEE INFOCOM*, Kobe, Japan, Apr. 1997.

[36] G. Maier, A. Feldmann, V. Paxson, and M. Allman. On dominant characteristics of residential broadband Internet traffic. In *Proc. Internet Measurement Conference*, Chicago, Illinois, Nov. 2009.

[37] G. Maier, F. Schneider, and A. Feldmann. NAT usage in residential broadband networks. In *Passive & Active Measurement (PAM)*, Atlanta, GA, Mar. 2011.

[38] MaxMind. http://www.maxmind.com/.

[39] J. Mogul, B. Krishnamurthy, F. Douglis, A. Feldmann, Y. Goland, A. van Hoff, and D. Hellerstein. *Delta encoding in HTTP*, January 2002. RFC 3229.

[40] J. C. Mogul, Y. M. Chan, and T. Kelly. Design, implementation, and evaluation of duplicate transfer detection in HTTP. In *Proc. 1st USENIX NSDI*, San Francisco, CA, Mar. 2004.

[41] Microsoft Smooth Streaming. http://www.iis.net/download/SmoothStreaming/.

[42] A. Muthitacharoen, B. Chen, and D. Mazieres. A low-bandwidth network file system. In *Proc. 18th ACM Symposium on Operating Systems Principles (SOSP)*, Banff, Canada, Oct. 2001.

[43] K. Park, S. Ihm, M. Bowman, and V. S. Pai. Supporting practical content-addressable caching with CZIP compression. In *Proc. USENIX Annual Technical Conference*, Santa Clara, CA, June 2007.

[44] K. Park, V. S. Pai, K.-W. Lee, and S. Calo. Securing Web service by automatic robot detection. In *Proc. USENIX Annual Technical Conference*, Boston, MA, June 2006.

[45] PlanetLab. http://www.planet-lab.org/, 2008.

[46] S. Podlipnig and L. Böszörmenyi. A survey of Web cache replacement strategies. *ACM Computing Surveys*, 35, Dec. 2003.

[47] L. Popa, A. Ghodsi, and I. Stoica. HTTP as the Narrow Waist of the Future Internet. In *Proc. 9th ACM Workshop on Hot Topics in Networks (Hotnets-IX)*, Monterey, CA, Oct. 2010.

[48] M. O. Rabin. Fingerprinting by random polynomials. Technical Report TR-15-81, Center for Research in Computing Technology, Harvard University, 1981.

[49] S. C. Rhea, K. Liang, and E. Brewer. Value-based Web caching. In *Proc. Twelfth International World Wide Web Conference*, Budapest, Hungary, May 2003.

[50] Riverbed Technologies, Inc. http://www.riverbed.com/.

[51] Roundup on Parallel Connections. http://www.stevesouders.com/blog/2008/03/20/roundup-on-parallel-connections/.

[52] RSS 2.0 Specification. http://www.rssboard.org/rss-specification.

[53] F. Schneider, S. Agarwal, T. Alpcan, and A. Feldmann. The new Web: Characterizing Ajax traffic. In *Passive & Active Measurement (PAM)*, Cleveland, OH, Apr. 2008.

[54] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. *RTP: A Transport Protocol for Real-Time Applications*. Internet Engineering Task Force, Jan. 1996. RFC 1889.

[55] H. Schulzrinne, A. Rao, and R. Lanphier. *Real Time Streaming Protocol (RTSP)*. Internet Engineering Task Force, Apr. 1998. RFC 2326.

[56] F. D. Smith, F. H. Campos, K. Jeffay, and D. Ott. What TCP/IP protocol headers can tell us about the Web. In *Proc. ACM SIGMETRICS*, Cambridge, MA, June 2001.

[57] N. Spring and D. Wetherall. A protocol-independent technique for eliminating redundant network traffic. In *Proc. ACM SIGCOMM*, Stockholm, Sweden, Sep. 2000.

[58] Squid Configuration Directive. http://www.squid-cache.org/Doc/config/quick_abort_min/.

[59] P. Srisuresh and K. Egevang. *Traditional IP Network Address Translator (Traditional NAT)*. United States, 2001. RFC 3022.

[60] StatCounter. http://statcounter.com/.

[61] W3Counter. http://www.w3counter.com/.

[62] L. Wang, K. Park, R. Pang, V. S. Pai, and L. Peterson. Reliability and security in the CoDeeN content distribution network. In *Proc. USENIX Annual Technical Conference*, Boston, MA, June 2004.

[63] 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. In *Proc. 17th ACM Symposium on Operating Systems Principles (SOSP)*, Kiawah Island, SC, Dec. 1999.