Understanding and Improving Modern Web Traffic Caching

Sunghwan Ihm

A Dissertation
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance
by the Department of
Computer Science
Adviser: Vivek S. Pai

September 2011

Abstract

The World Wide Web is one of the most popular and important Internet applications, and our daily lives heavily rely on it. Despite its importance, the current Web access is still limited for two reasons: (1) the Web has changed and grown significantly as social networking, video streaming, and file hosting sites have become popular, requiring more and more bandwidth, and (2) the need for Web access also has grown, and many users in bandwidth-limited environments, such as people in the developing world or mobile device users, still suffer from poor Web access.

There was a burst of research a decade ago aimed at understanding the nature of Web traffic and thus improving Web access, but unfortunately, it has dropped off just as the Web has changed significantly. As a result, we have little understanding of the underlying nature of today's Web traffic, and thus miss traffic optimization opportunities for improving Web access. To help improve Web access, this dissertation attempts to fill the missing gap between previous research and today's Web.

For a better understanding of today's Web traffic, we first analyze five years (2006-2010) of real Web traffic from a globally-distributed proxy system, which captures the browsing behavior of over 70,000 users from 187 countries. Using this data set, we examine major changes in Web traffic characteristics that occurred during this period. We also develop a new Web page analysis technique that is better suited for modern Web page interactions. Using our analysis technique, we analyze various aspects of page-level changes, and present a simple Web traffic model that we develop based on our findings. Finally, we investigate the redundancy of this traffic, using both traditional object-level caching as well as content-based approaches that use the caching technique at the sub-object or packet level. Among many findings, we observe a huge potential benefit of the content-based caching approaches - the byte hit rate is almost twice as large as that of the traditional object-level caching approach.

Motivated by the possible benefits from content-based caching approaches, we also develop Wanax, a scalable and flexible wide-area network (WAN) accelerator that is designed for low-bandwidth and resource-limited developing world environments. It uses a novel multi-resolution chunking (MRC) scheme that provides high compression rates and high disk performance for a variety of content, while using much less memory than existing approaches. Wanax exploits the design of MRC to perform intelligent load shedding to maximize throughput even when running on resource-limited shared platforms. Finally, Wanax exploits mesh network environments, instead of just the star topologies common in enterprise branch offices. Equally importantly, the designs of Wanax can be applied to enterprise environments, providing the same benefits.

Acknowledgements

This dissertation would not have been possible without the help and support from many people. First of all, I owe my deepest gratitude to my advisor Vivek Pai. He has guided me throughout my graduate program. Without his continuous support, patience, enthusiasm, inspiration, and deep knowledge, I would not be where I am today. I also would like to thank my thesis committee members Larry Peterson, Jennifer Rexford, Mike Freedman, and Andrea LaPaugh for the useful feedback that greatly improved the quality of this dissertation.

Many people in the department have made available their support in a number of ways. The first half of this dissertation could not have been possible without the support from the CS Department Staff. In particular, I would like to thank Scott Karlin, Chris Tengi, Chris Miller, and Paul Lawson for helping me set up and run large-scale data analysis jobs on the department cluster machines. For the second half of this dissertation, I am grateful to Marc Fiuczynski for arranging and coordinating the deployment of the Wanax system in Africa. I would like to give my special thanks to the Graduate Coordinator, Melissa Lawson. Her help was indispensable throughout my graduate program, and she also helped improve my language skills.

I was extremely lucky to have two research internship opportunities outside campus. I would like to thank T. V. Lakshman, Sarit Mukherjee, and Liming Wang at Alcatel-Lucent Bell Labs, and Byoung-Gon Chun, and Petros Maniatis at Intel Labs Berkeley for their collaboration in diverse research projects.

I would like to show my gratitude to my colleagues and friends at Princeton. I thank KyoungSoo Park, Anirudh Badam, Wonho Kim, and Changhoon Kim, and other members of network systems group for the useful discussion and feedback. I am grateful to my host family from the Rotary Foundation, Knud and Lindsey Christiansen for helping me settle down and learn the culture of the United States.

I also would like to thank Kuk Jin Jang for proofreading this dissertation, and my roommate Yoosik Kim.

Last but not least, I would like to thank my father, mother, brother, and sister who have supported and encouraged me throughout my graduate study; I dedicate this dissertation to them. Finally, I could not imagine being able to finish this program without Minkyung; I thank her for the endless support.

This dissertation was partly supported by the Rotary Foundation Ambassadorial Scholarship, NSF awards CNS-0615237, and CNS-0916204. The content of this dissertation was previously published in conference proceedings [40, 41, 43].

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The World Wide Web is one of the most popular and important Internet applications, and many people's daily activities heavily rely on it. For example, we read news, send/receive emails, search, shop, and watch video through the Web. Due to the importance of the Web in our daily lives, it is thus very critical for people to have good Web access. Indeed, the United Nations has recently proposed that Internet access should be a human right [34].

Despite its importance, current Web access is still limited for two reasons. First, since its first appearance in 1991, the Web has changed and grown significantly, requiring more and more bandwidth. While peer-to-peer application traffic dominated the Internet for some time, the volume of the Web traffic is increasing and is once again beginning to dominate Internet traffic [44, 49, 53]. The main driver is the popularity of social networking (*e.g.*, Facebook), file sharing (*e.g.*, RapidShare), and video streaming (*e.g.*, YouTube) sites. These changes in 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 [76], but also as many existing applications, such as game and video, are consolidated to the Web [49].

Second, the need for Web access also has grown, exposing the poor Web access problem for many users in bandwidth-limited environments. For example, billions of people in the developing world have very limited Web access due to poor network infrastructures. Even in the developed countries, the problem remains for people who access the Web via cellular or wireless networks. As the deployment of mobile devices becomes increasingly widespread, the problem of poor Web access will be exacerbated. Aside from mobile users, 5-10% of people in the United States do not have the bandwidth that allows basic Web functions, such as downloading images [66].

In order to improve Web access, we must first understand the underlying characteristics of Web traffic. Web traffic results from interactions among many components including clients, proxies, and servers, so only a detailed understanding of these interactions can offer insight for improving Web access. 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. Also, investigating the Web page loading process reveals where the bottleneck lies in the user-perceived page loading latency. Furthermore, analyzing the redundancy of Web traffic and effectiveness of caching could shape the design of caching mechanisms.

1.1 Previous Studies and Limitations

There was a burst of research a decade ago aimed at understanding the nature of Web traffic and thus improving Web access. Unfortunately, the research has dropped off just as the Web has changed significantly. As a result, we have little understanding of the underlying characteristics of today's Web traffic, and thus miss traffic optimization opportunities for improving Web access. We group the previous studies into the following three categories, and discuss their limitations and challenges.

1.1.1 Web Traffic Analysis and Modeling

A common approach for empirically modeling Web traffic is to reconstruct end-user browsing behavior from access log data which records accesses to Web pages. It assumes simple and static Web pages, and once Web pages are identified, other relevant model parameters can be derived, such as the number of embedded objects, total page size, total page time, and inter-arrival time. Many previous studies on analyzing and modeling Web traffic have used this approach [10, 52, 93].

However, there are two limitations with this approach. First, as the Web has changed significantly, the previous analysis approach has become inappropriate for today's dynamic Web pages that involve complex client-side interactions (*e.g.*, Ajax [22]). When applied to today's Web traffic, the resulting traffic model would be inaccurate. For a better understanding of modern Web page interactions, we need a new Web page analysis and modeling technique. Second, many studies have analyzed relatively small-scale data sets that are also time and location specific (*e.g.*, one month of a certain university or enterprise Web proxy trace). For tracking and understanding changes in Web traffic, we need large-scale data sets spanning a multi-year period, collected under the same conditions.

1.1.2 Object-based Caching

Conventional object-based caching works by storing previously seen Web objects near clients (*e.g.*, browsers or proxies), and serving them locally for future requests [31]. It can reduce network traffic, server overhead, and user-perceived latency. Many researchers have heavily studied the implications of object-based caching [12, 13, 36, 60, 106, 106].

Unfortunately, object-based caching has two critical limitations that diminish its utility on modern Web traffic. First, it operates on a whole-object basis, and thus is effective only for identical objects with the same URLs. For example, even if

the content of two objects is 99% identical, object-based caching cannot exploit this redundancy. If two identical objects have different URLs (aliasing [47]), object-based caching considers them as two different objects. Second, it is effective only for those objects designated as cacheable, such as static text and images. However, many uncacheable objects can be in fact similar or even identical. Object-based caching misses this traffic optimization opportunity.

1.1.3 Content-based Caching

Content-based caching works by splitting an object or file into many smaller chunks, and caching those chunks instead of entire objects [94]. Chunk boundaries are determined based on the content, commonly with Rabin's fingerprinting [80]. Unlike offset-based chunk segmentation schemes (*e.g.*, every 1 KB), any insertion/deletion/modification to the content only affects nearby chunks. Once chunk boundaries are detected, chunks are named based on the content, often with SHA-1 hash [65]. The next time the system sees the same chunk, it can pass only a reference instead of the original content. Since its first proposal, many systems such as network file systems [7, 64], WAN acceleration [84], Web caching [70, 82], and storage systems [24] have used this approach. When applied to Web traffic, content-based caching can yield higher cache hit rates than the object-based caching, as it can eliminate redundancy within objects and across different objects. Furthermore, it is protocol independent and thus effective for uncacheable content as well.

Despite the potential benefits of content-based caching, it is still not fully understood in several ways. First, while the implications of object-based caching approaches have been heavily studied in the past, there is a lack of a detailed understanding of content-based caching on real Web traffic. For example, it is unclear how much additional benefit content-based caching approaches can provide compared to the use of the object-based caching, and where the benefits arise. Also, estimating the re-

quired cache storage size would help plan the deployment of content-based caching systems. Second, content-based caching systems can outperform object-based caching systems at the cost of more resources such as large main memory and fast disk. While bandwidth-limited developing world environments would benefit most from content-based caching systems, how to design such systems that can run in resource-limited environments remains an open question.

1.2 Our Approach and Contributions

To summarize, this dissertation attempts to answer the following questions.

- What has changed in Web traffic over time?
- How can we analyze today's dynamic Web pages that involve complex client-side interactions such as Ajax?
- What is the implication of content-based caching on today's real Web traffic, such as the effective byte hit rate and required cache storage size?
- How can we design a content-based caching system that is suitable for resource-limited developing world environments?

To answer the first three questions, in Chapter 2, we analyze large-scale real Web traffic data from a globally-distributed proxy system [40, 41]. Motivated by the potential benefits of content-based caching approaches from our analysis, in Chapter 3, we address the last question by presenting Wanax, a scalable and flexible wide-area network (WAN) accelerator that is designed for low-bandwidth and resource-limited developing world environments [43].

1.2.1 Understanding Modern Web Traffic

A better understanding of today's Web traffic involves several challenges. First, tracking changes over time requires large-scale data sets spanning many years, 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, generating inaccurate analysis results. Third, investigating the implications of content-based caching approaches requires full content data rather than just access logs.

To overcome these challenges, in Chapter 2, 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 technique that is better suited for modern Web page interactions by grouping requests into streams and exploiting the structure of Web pages. Using this analysis technique, we analyze various aspects of page-level changes, and also present a simple Web traffic model based on our findings. Finally, we capture the full content of traffic, and investigate the redundancy of this traffic, using both traditional object-level caching as well as content-based approaches.

Among many findings, we observe a rise of Ajax and Flash video (FLV). For example, we see a consistent increase of the traffic volume and object size of JavaScript, CSS [23], advertising networks, and video sites' traffic over time. Interestingly, the most accessed (directly or indirectly) site by users is either a search engine or an analytics site. They reach an increasingly large fraction (65%) of the user population, which has implications for user tracking and privacy.

In addition, our new Web page analysis technique 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, 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 per browser and the improved caching behavior of Web sites.

Finally, 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 in the content between different versions of an object as well as redundancy across different objects. Furthermore, a small number of aborted requests (1.8-3.1%), mostly video, can negatively impact object-based caching performance because their volume would comprise a significant portion of the entire traffic (69.9-88.8%) if they were fully downloaded.

1.2.2 Improving Modern Web Traffic Caching

Wide-area network (WAN) accelerators operate by compressing redundant network traffic from point-to-point communications, enabling higher effective bandwidth. Unfortunately, while network bandwidth is scarce and expensive in the developing world, current WAN accelerators are designed for enterprise use, and are a poor fit in these environments. More specifically, limited resources in the developing world, such as a small amount of main memory and slow commodity disks, severely degrade WAN acceleration performance.

In Chapter 3, we present Wanax, a WAN accelerator designed for developing-world deployments. It uses a novel multi-resolution chunking (MRC) scheme that provides high compression rates and high disk performance for a variety of content, while using much less memory than existing approaches. Wanax exploits the design of MRC to perform intelligent load shedding to maximize throughput when running on resource-

limited shared platforms. Finally, Wanax exploits the mesh network environments being deployed in the developing world, instead of just the star topologies common in enterprise branch offices. Combining these approaches, Wanax provides scalable and flexible WAN acceleration for the developing regions.

We also implement a prototype of Wanax, and demonstrate that it not only enables smooth video streaming, but also reduces response times of Web browsing in developing world environments. Furthermore, when equipped with high-end SCSI disks, Wanax offers comparable performance to enterprise WAN accelerators, with a much smaller memory footprint.

1.3 Dissertation Overview

This dissertation is organized as follows: in Chapter 1, we motivate the problem of limited Web access, and also discuss background and limitations of previous studies for improving Web access. For a better understanding of today's Web traffic, Chapter 2 presents a detailed analysis of modern Web traffic using the CoDeeN content distribution network data set. Based on the analysis results, Chapter 3 describes how we design and implement Wanax, which enables high performance WAN acceleration in resource-limited environments. Finally, we conclude and discuss future work in Chapter 4.

Chapter 2

Towards Understanding Modern Web Traffic

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 [44]. 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 [76], but also as applications get migrated to the Web [49].

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 [43,82,84] 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 plan their deployment.

While much research activity occurred a decade ago aimed at better understanding the nature of Web traffic [10, 12, 52, 93, 106], 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 [22]). Third, understanding the effectiveness of content-based caching approaches requires full content data rather than just access logs.

In this chapter, 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. Using this algorithm, we analyze various aspects of page-level changes, and also present a simple Web traffic model that we developed based on our findings. 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 (Section 2.2) One of the major changes in modern Web traffic is a rise of Ajax and Flash video (FLV) usage – we see a consistent increase of its traffic volume as well as the object size of JavaScript and CSS [23], and traffic to advertising networks and video sites. The maximum number of concurrent connections also has increased due to browser changes for accommodating Ajax. Surprisingly, the most accessed site by users is either a search engine or an analytics site. They reach an increasingly large fraction (65%) of the user population, which

has implications for user tracking and privacy. In addition, we observe clear regional differences in client bandwidth, browser popularity, and dominant content types that need to be considered when designing/deploying systems. Finally, we observe an increase of Network Address Translation (NAT) [96] usage, which is likely related to the scarcity of IPv4 addresses.

Page-Level Characteristics (Section 2.3) 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, based on our findings, we provide a simple Web traffic model that represents modern Web traffic.

Redundancy and Caching (Section 2.4) We find two interesting trends in URL popularity: 1) the popular URLs become more popular, which potentially improves caching, but 2) the long tail of the content is also growing, which can 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 the redundancy across different objects. Finally, a small number of aborted requests (1.8-3.1%), mostly video, can

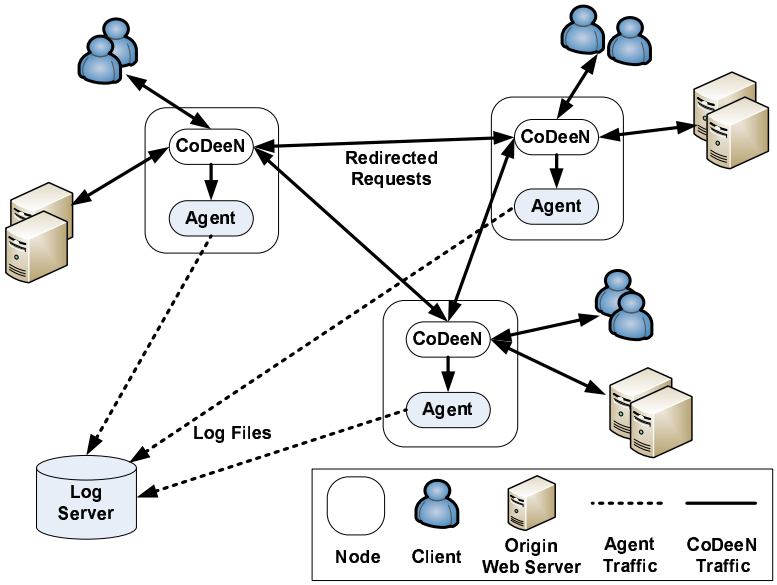


Figure 2.1: Data collection

negatively impact object-based caching performance because the volume of those target objects would comprise a significant portion of all traffic (69.9-88.8%) if they were fully downloaded.

The rest of this chapter is organized as follows: in Section 2.1, we describe the details of our data set. Section 2.2 examines the major changes in high-level characteristics of Web traffic. Section 2.3 presents the detailed page-level analysis with our new page detection algorithm, and Section 2.4 analyzes redundancy and caching. Finally, we discuss related work in Section 2.5, and conclude in Section 2.6.

2.1 Data Set

We use traffic from the CoDeeN content distribution network (CDN) [103], 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 [73] nodes. The term "semi-open" means that while anyone can use CoDeeN by configuring his or her browser, it only allows GET requests and bans other methods such as CONNECT,

PUT, or POST for security reasons. 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 [71].

Our data set consists of two parts. First, CoDeeN records all requests not served from the client's browser cache in the Squid-style log format. [1] We use these access logs for examining any longitudinal changes in Section 2.2, 2.3, and 2.4. In addition, we capture the full content of the cache-miss traffic between the CoDeeN nodes and the origin Web servers by running an agent process on each node as shown in Figure 2.1. Using this full content data, we evaluate both object-based and content-based caching approaches, and analyze aborted transfers in Section 2.4.

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 consists of about 280-460 million requests from 240-360 thousand unique client IPs. The clients IPs originate from 168-187 countries and regions as determined using the MaxMind database [57], and cover 40-60% of /8 networks, and 7-24% of /16 networks. The total number of unique servers ranges from 820 thousand to 1.2 million.

Our large-scale data set spanning many years is one of the largest Web traffic data sets collected. Also, it is potentially more representative than many other available data sets because its world-wide client population is not skewed toward a specific organization, such as a university or company [6, 106].

[1] Timestamp, service time, request URL, method, user agent, content type, referer, response code, and response size.

USA	2006	2007	2008	2009	2010
Requests (K)	33457	40306	24489	23242	14425
Volume (GB)	391	627	338	316	261
# IPs	19122	21767	13590	13316	12938
# Agents	23323	27021	17574	16898	16720
China	2006	2007	2008	2009	2010
Requests (K)	22455	88754	29918	38108	22871
Volume (GB)	394	1177	404	409	278
# IPs	49296	94892	38818	43234	33365
# Agents	53877	109651	45132	51845	41933
France	2006	2007	2008	2009	2010
Requests (K)	2207	3949	3255	3550	3322
Volume (GB)	21	45	33	42	50
# IPs	3619	5061	3197	3719	5111
# Agents	3933	5514	3481	4293	5970
Brazil	2006	2007	2008	2009	2010
Requests (K)	1493	4452	2043	3923	7123
Volume (GB)	16	54	22	44	100
# IPs	1417	8639	3261	3070	9536
# Agents	1630	9986	3776	3626	10925
Total	2006	2007	2008	2009	2010
Requests (K)	59612	137461	59705	68823	47741
Volume (GB)	822	1903	797	811	689
# IPs	73454	130359	58866	63339	60950
# Agents	82763	152172	69963	76662	75548

Table 2.1: Summary statistics for captured access logs, sampled one month (April) per year.

For the remainder of the chapter, 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 2.1 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 analy-

sis of four countries covers 48-137 million requests, 689-1903 GB traffic, and 70-152 thousand users per month.

2.2 High-Level Characteristics

In this section, we analyze high-level characteristics of our Web traffic data, looking at the properties of clients in Section 2.2.1, and objects and domains in Section 2.2.2.

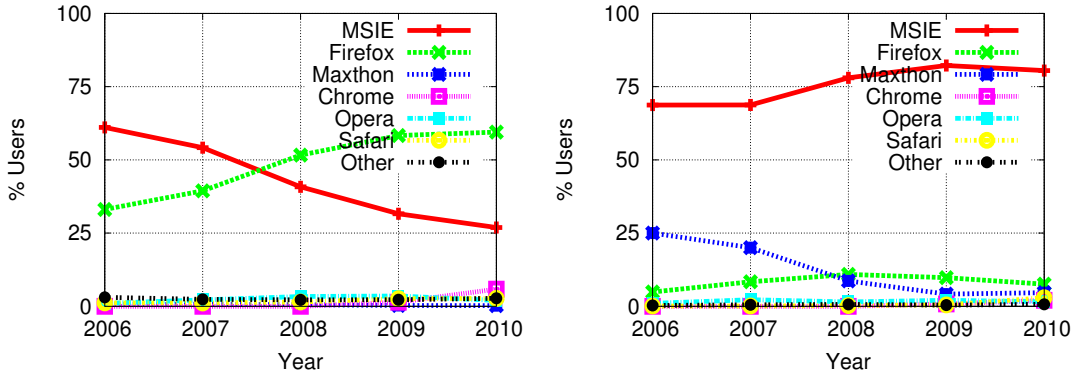
The summary of our findings are as follows:

- One of the major changes is a rise of Ajax and Flash video (FLV) usage – its traffic volume is steadily increasing, and also the object size of JavaScript and CSS has grown over time. FLV object size is larger than that of other video types.

- We also observe an increase of the maximum number of concurrent connections. This is due to browsers increasing the default number of maximum connections, in order to accommodate Ajax which usually requires many simultaneous connections for low latency.

- In terms of domain names, we see a consistent increase of video and advertising network/analytics sites' traffic during the five-year period. This is also related to the rise of Ajax and FLV. The single top site accessed by the most users is either a search engine or an analytics site. It reaches an increasingly large fraction (65%) of the user population, which has implications for user tracking and privacy.

- While we find many consistent changes across different countries, we also observe clear regional differences in client bandwidth, browser popularity, and dominant content types. One needs to consider these differences when designing/deploying network systems for these regions.



(a) US (b) CN

(c) FR (d) BR

Figure 2.2: Browser popularity: Firefox and MSIE together account for more than 86% of the entire users. MSIE dominates in China.

- Finally, we observe an increase of NAT usage, which is likely related to the scarcity of IPv4 addresses and the common use of browser plug-ins.

2.2.1 Clients

In this section, we analyze the changes in browser popularity, connection speed, NAT usage, and maximum number of concurrent connections.

Browser Popularity We first investigate browser popularity by looking at the `User-Agent` field in the access log. Understanding which browser is dominant is important because browsers behave differently under the hood, which affects the

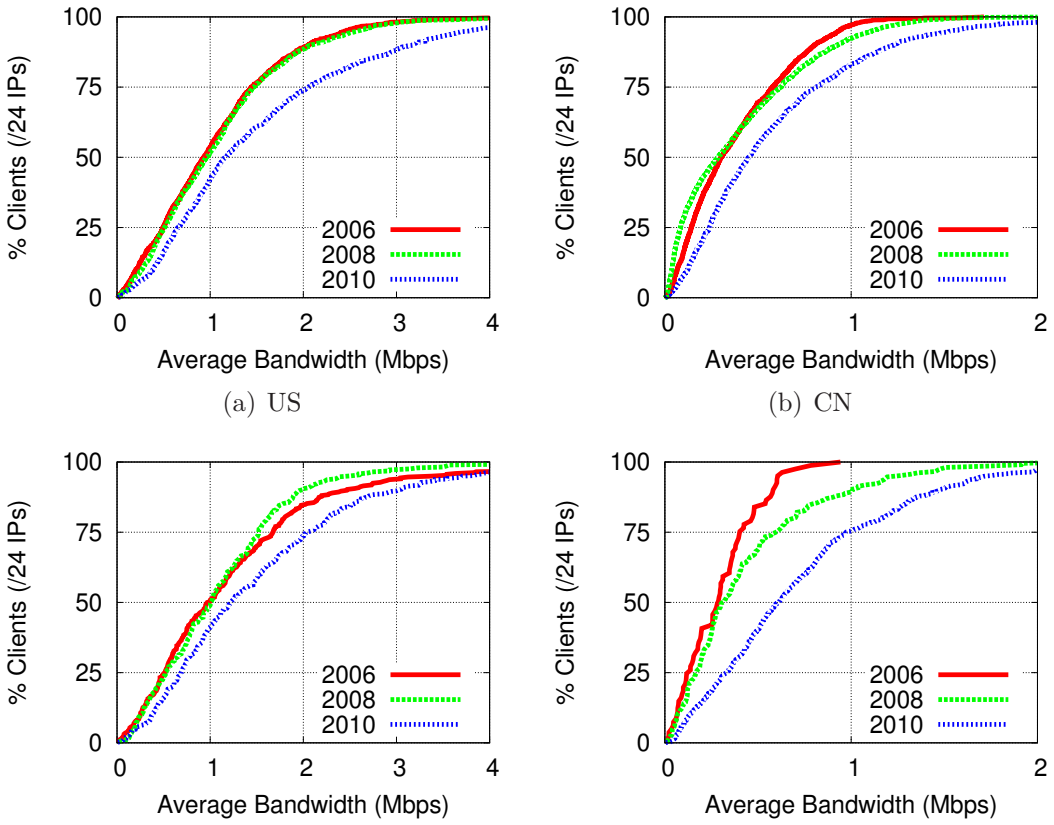
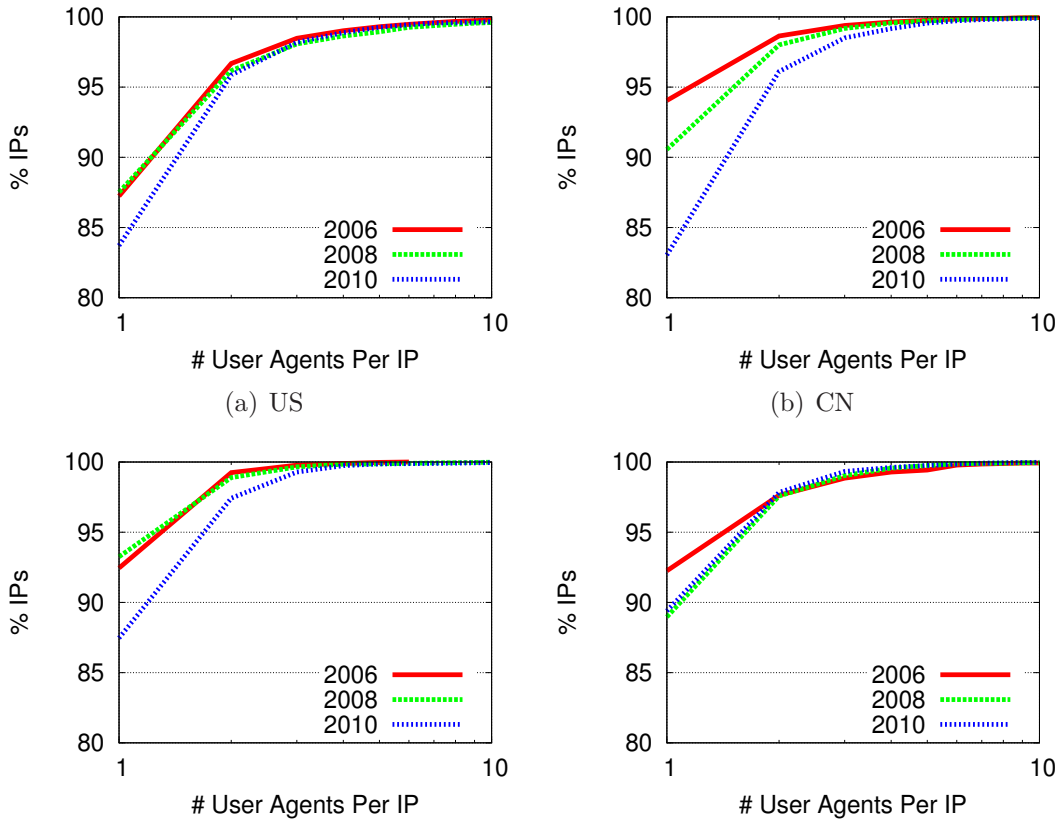


Figure 2.3: Average client bandwidth: Client bandwidth improves over time.

resulting Web traffic. For example, browsers have different default values for the maximum number of concurrent connections and browser cache sizes.

Figure 2.2 shows the percentage of different browsers in terms of the number of users. 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 countries. The popularity of Firefox consistently increases, exceeding the share of MSIE by 2007 and 2008. We also observe a slowly increasing share of Chrome in the United States and Brazil. Interestingly, MSIE remains the dominant browser in China, and Maxthon – one of the most popular browsers in China – is losing its popularity. Considering that Maxthon is based on a customized MSIE, Maxthon and MSIE together account for more than 85% of the user population in China while Firefox represents only 5-11%.



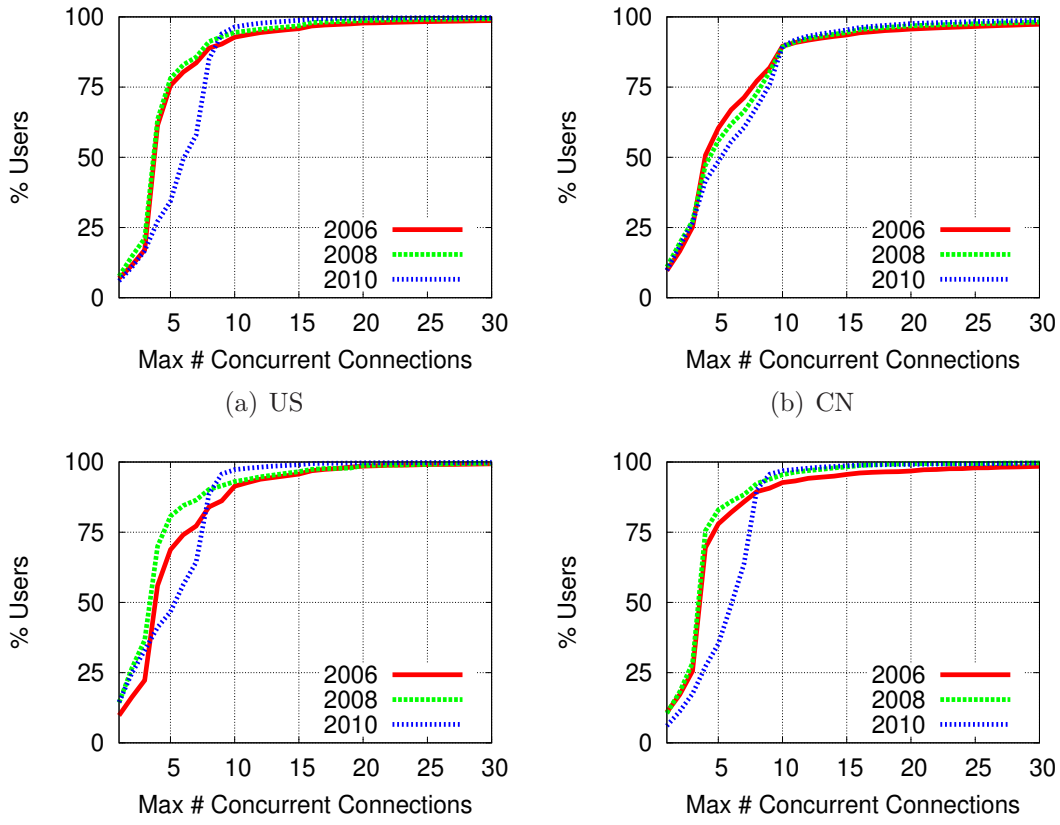
(a) US (b) CN

(c) FR (d) BR

Figure 2.4: NAT usage: Most (83-94%) client IPs have only one user agent. The number has grown slightly over time due to the increasing use of NATs or browser plug-ins.

Connection Speed We estimate the client bandwidth by observing the download time for objects. To minimize the effect of link latency, we consider only those objects that are larger than 1 MB. Figure 2.3 shows CDFs of average client bandwidth per aggregated /24 IP address. Overall, we observe that the client bandwidth is consistently increasing over time. [2] 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.

[2]PlanetLab connectivity did not have a major improvement during the five-year period.



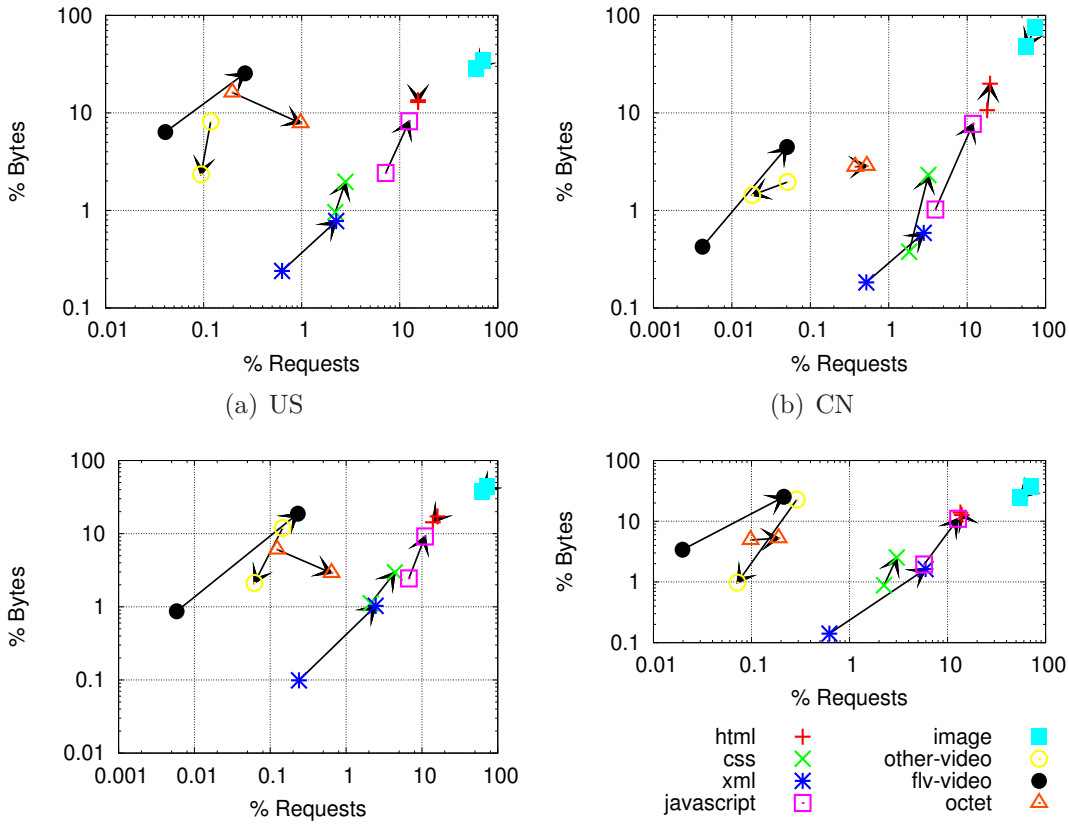
(a) US (b) CN

(c) FR (d) BR

Figure 2.5: Maximum number of concurrent connections per user: We observe a large increase in 2010 due to the browsers increasing the number of connections.

NAT Usage We analyze the use of NAT in Figure 2.4 where we present CDFs of the number of different user agents per client IP address. While most (83-94%) client IPs have just one user agent, the number has grown slightly over time. This increase may be related to the scarcity of IPv4 addresses, or the common use of browser plug-ins or toolbars that have its own user agent string. The maximum number of user agents per IP we observe is 69 in the United States, 500 in China, 36 in Brazil, and 20 in France.

Maximum Concurrent Connections Figure 2.5 shows CDFs of the maximum number of concurrent connections per user agent. We observe a large increase in 2010 – the median number grows from 4-5 in 2006 and 2008 to 6-7 in 2010. This



(a) US (b) CN

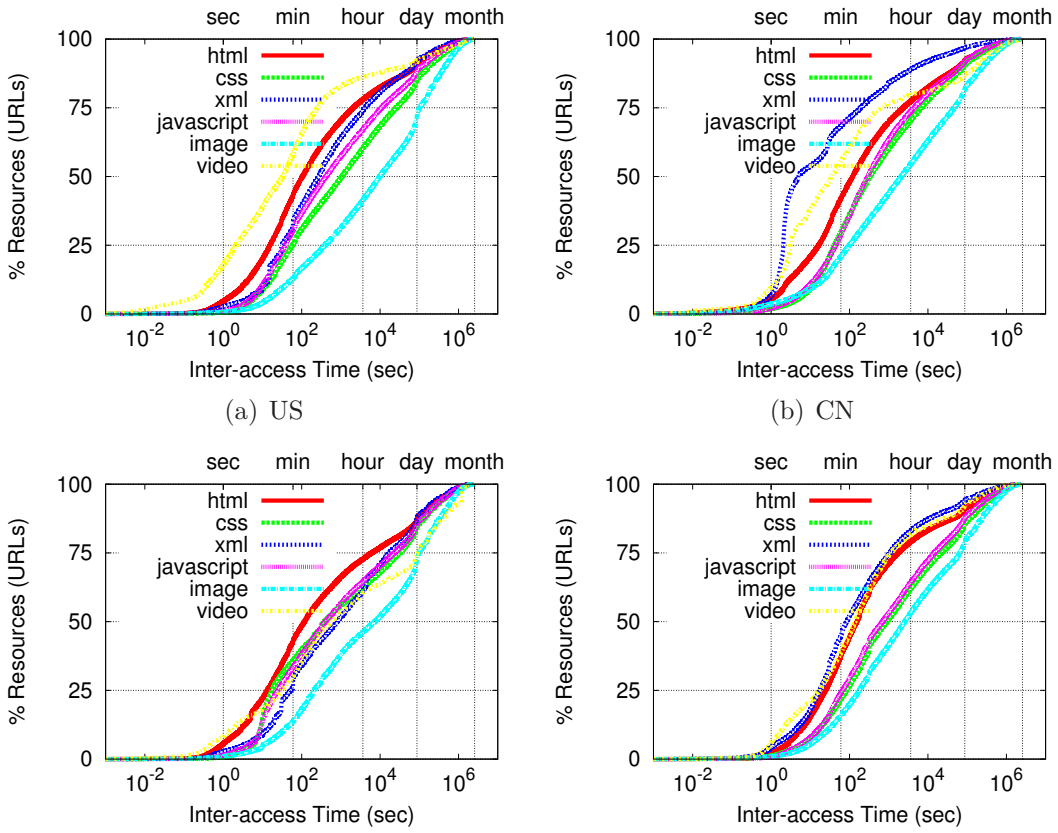
(c) FR (d) BR

Figure 2.6: Content type distribution changes from 2006 to 2010: We observe a growth of Flash video, JavaScript, CSS, and XML usage. Images are still dominating request traffic.

increase is mainly because the browsers change the default number of maximum simultaneous connections per server from 4 to 6 in around 2008 [3, 86]. This is in order 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 [31], which is very different from the current usage.

2.2.2 Objects and Domains

In this section, we examine the changes in content type distribution, object size, and top sites in terms of requests and bytes.



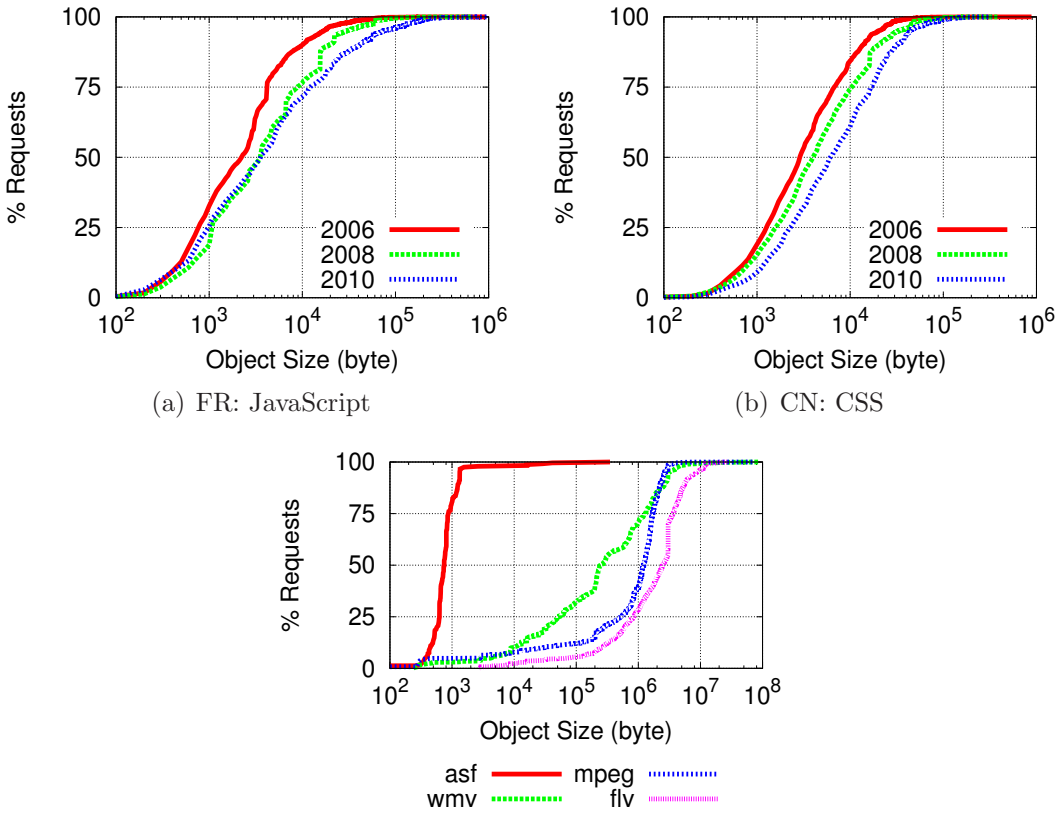
(a) US (b) CN

(c) FR (d) BR

Figure 2.7: Access rate in 2010: JavaScript, CSS, and images are served from the browser cache and not frequently accessed.

Content Type We observe a shift from static image-oriented Web pages to dynamic rich media Web pages in Figure 2.6. It presents the content type distribution changes Brazil 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 increased popularity of Ajax. We also find a sharp increase of Flash video (FLV) traffic, accounting for about 25% of the total traffic both in the United States and Brazil in 2010. At the same time, non-FLV traffic decreases, demonstrating the transition of the media delivery medium to the popular Flash video. In addition, while the byte percentage of octet-stream traffic sees a general decrease, its percentage of request traffic increases. This may be related to the custom use of HTTP as a



(a) FR: JavaScript

(b) CN: CSS

(c) US, 2010: video

Figure 2.8: Object size: The object size of JavaScript and CSS has increased. Flash video object size is bigger than that of other video.

transport protocol for exchanging binary data in many applications. Still, the 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 within the traffic patterns as shown in Figure 2.7 where we show CDFs of inter-access time for each unique URL in 2010, with different content types. 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. Figure 2.8 (a) presents CDFs of JavaScript sizes in France, and we show CDFs of CSS sizes in China in Figure 2.8 (b), from 2006 to 2010. These changes are likely related to the popular use of Ajax. In general, other content types do not show consistent changes.

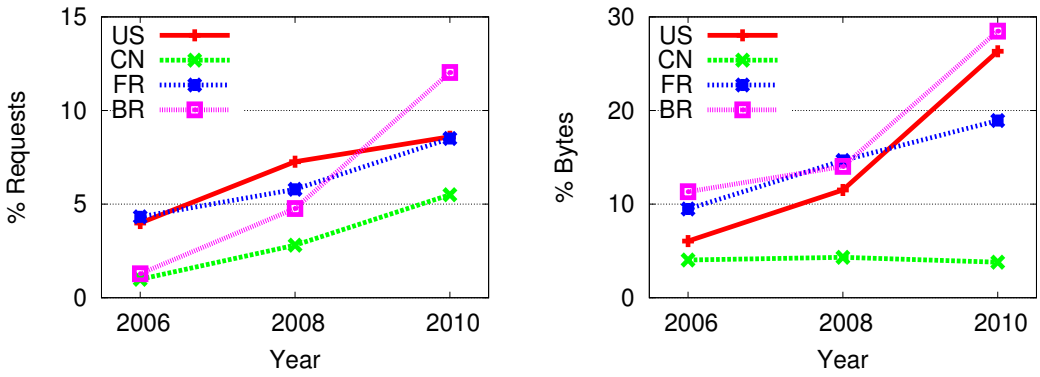
While there seems to be no significant size changes over time in video objects, we observe FLV is bigger than other video in general. Figure 2.8 (c) compares the object size (CDF) of different video types in the United States for 2010. Some video objects (*e.g.*, ASF) are very small in size and are container 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 [91] or RTP [90] 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 were introduced recently in late 2009 and 2010 [1, 8, 63], 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 as well as an increasing number of requests. We plan to analyze this case with a more recent data set in the future.

Domains We examine the share of 1) video site [3] traffic, and 2) advertising network/analytics [4] site traffic in Figure 2.9. We consider the top 50 sites that dominate these kinds of traffic. In Figure 2.9 (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 [46]. In addition, we find the volume of video site traffic is consistently

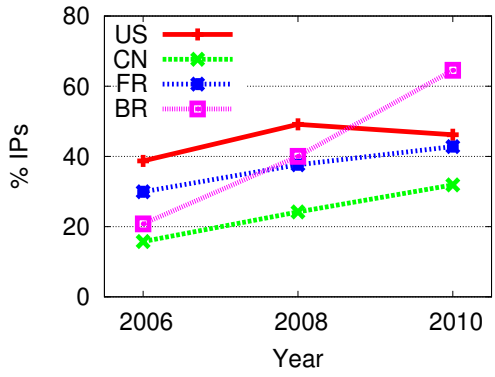
[3] *e.g.*, `youtube.com`

[4] *e.g.*, `doubleclick.com`, `google-analytics.com`



(a) Ads network traffic by requests

(b) Video site traffic by bytes



(c) Single top site % IPs

Figure 2.9: Top sites: Ads/video site traffic is increasing. A single top site tracks up to 65% of the user population.

increasing as shown in Figure 2.9 (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 2.9 (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 (`google-analytics.com`). Especially in 2010, the percentage reaches up to 65% in Brazil, which has implications for user tracking and privacy.

2.3 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 2.3.1. In Section 2.3.2, we present a new page detection algorithm called StreamStructure that is better suited for modern Web traffic analysis, and demonstrate it is more accurate than previous approaches. Using this algorithm, Section 2.3.3 examines the initial page characteristics, and analyzes page loading latency via simulations. Finally in Section 2.3.4, we present a simple Web traffic model that we developed based on our findings.

The summary of our findings are as follows:

- 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. In general, the pages become increasingly complex in that both the size and number of objects increase. On the other hand, we observe that page loading latency drops and the object inter-arrival rate becomes burstier in 2009 and 2010 due to the increased number of concurrent connections and improved caching behavior of Web sites.

- Using our simulator, we experiment with the effect of various tuning approaches. We find that increasing the number of simultaneous connections from the browser could further reduce page load latency by 23%. Removing dependencies between objects – some objects (*e.g.*, images) can be fetched only after some other objects (*e.g.*, HTML or JavaScript) are downloaded – would yield at most a 50% reduction. Reducing per-object latency by 50% would actually reduce page-load latency by 67% because of the simultaneous connections. Together, page loading latency can be reduced by up to 75%.

2.3.1 Page Detection Algorithm

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 [10, 52, 93]. 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 [18]. 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.

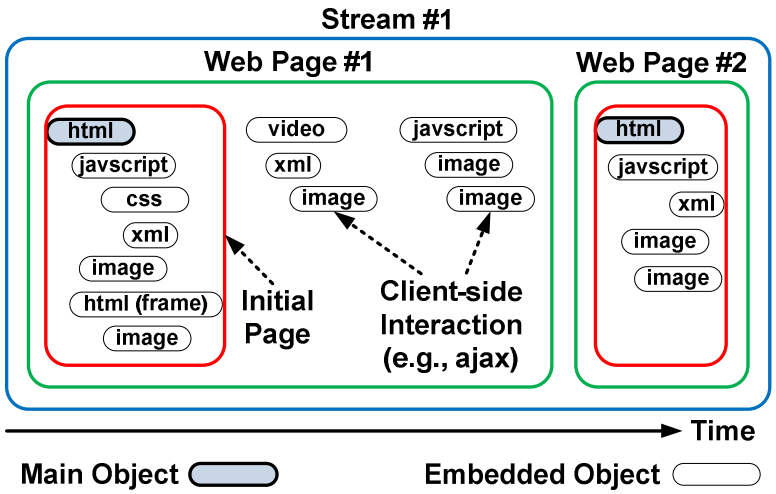


Figure 2.10: Streams, Web pages, initial pages, client-side interactions, and main/embedded objects.

2.3.2 StreamStructure Algorithm

To overcome the limitations of the previous approaches, we develop a new page detection algorithm called StreamStructure, that is more accurate than the previous approaches. StreamStructure exploits the stream and structure information of Web pages, and consists of three steps – grouping streams, detecting main objects, and identifying initial pages. Figure 2.10 depicts the definition of streams, Web pages, initial pages, main and 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 a 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 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 in Figure 2.2. 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 the 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 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 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 [4] with MSIE. We visit approximately ten Web pages for each site, resulting 1,197 Web pages in total. [5] We also record the URLs of those visited

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

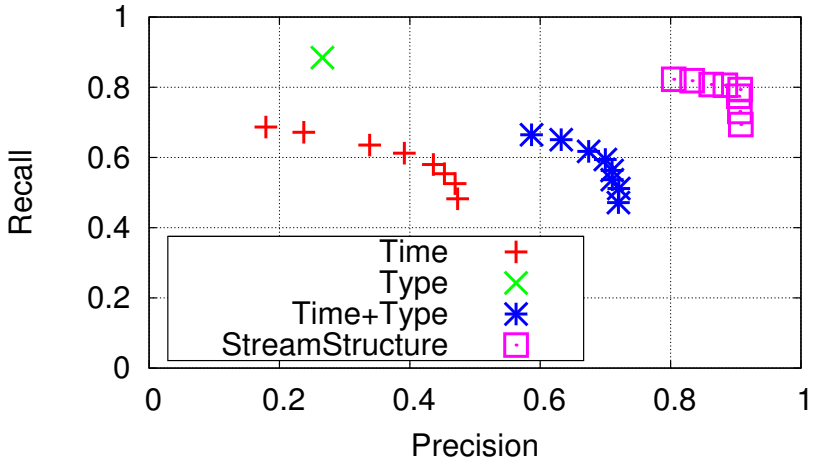
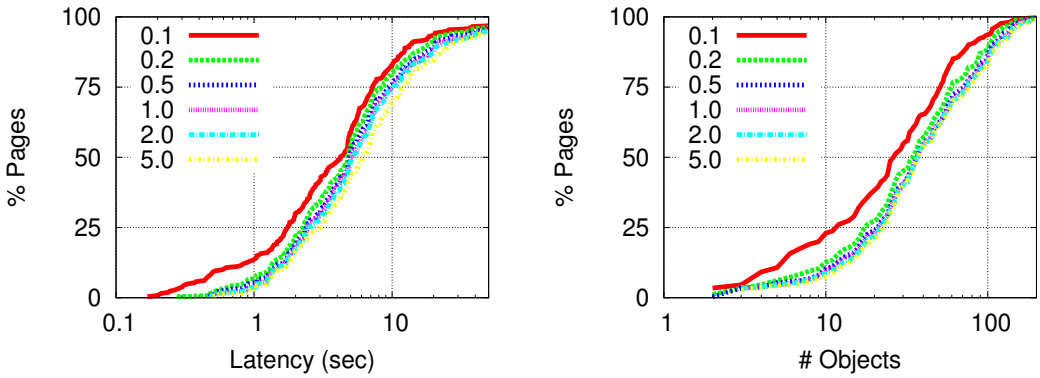


Figure 2.11: Precision and recall: StreamStructure outperforms previous page detection algorithms, simultaneously achieving high precision and recall. It is also robust to the idle time parameter selection.

Web pages (or main objects), and compare them with the URLs of the Web pages found by each approach.

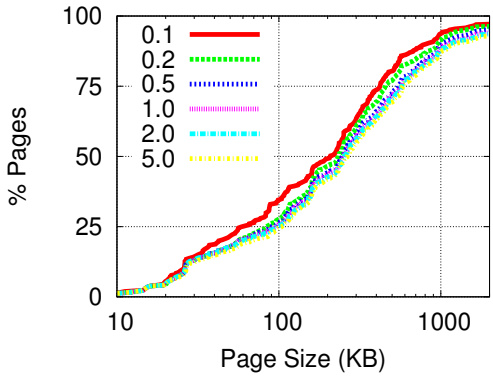
Figure 2.11 shows the precision and recall of various approaches. Precision is defined as the number of correct Web pages found divided by the total number of Web pages found, and recall is defined as the number of correct Web pages found divided by the total number of correct Web pages. For comparison, we also try a simple combination of time-based and type-based approaches (Time+Type) that do 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



(a) Page loading time (sec)

(b) # Objects per page



(c) Page size (KB)

Figure 2.12: Initial page idle time parameter selection: The idle time between 0.5 and 2 seconds provides stable results.

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 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 as in Figure 2.12. Overall, 23.9% of pages have Google Analytics beacons in our manually collected 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.

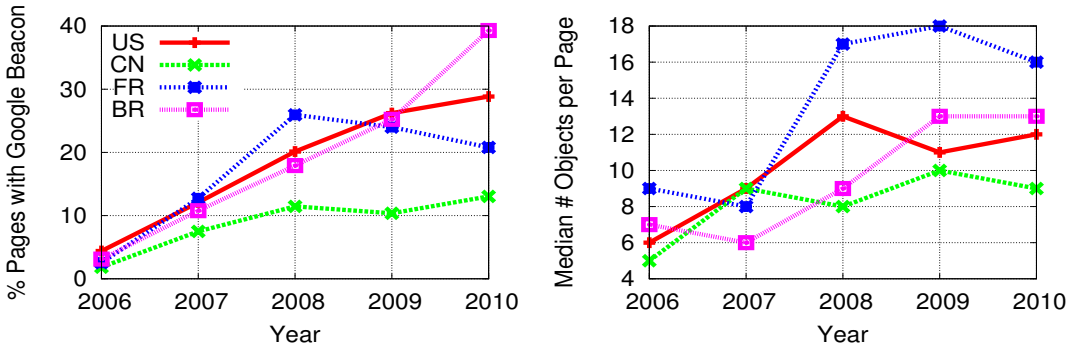
2.3.3 Analysis Results

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.

Initial page characteristics We first show the fraction of Web pages that have a Google Analytics beacon in our data set in Figure 2.13 (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 a very significant amount.

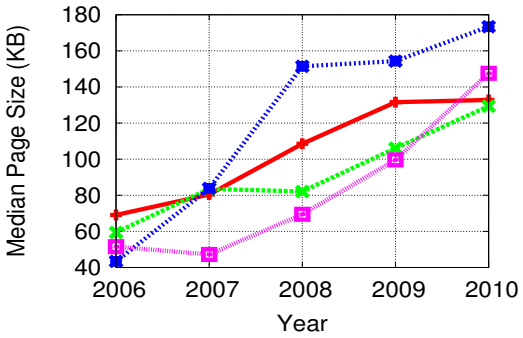
In Figure 2.13 (b)-(f), we examine the changes in the number of objects, size, and latency of the initial pages, and the latency and inter-arrival time of each individual object in the initial pages. We compare the median values rather than the mean, since our page detection algorithm is not perfect and the mean can be greatly biased by outliers/errors.

We observe a consistent increase of the number of objects and the total size in Figure 2.13 (b) and (c), indicating that the pages have become increasingly complex. 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 has become bigger from 69 KB in 2006 to 133 KB in 2010. This increase must be related to the popular use of advertisement/analytics, and the object size increase of JavaScript and CSS in Figure 2.8 (a) and (b).

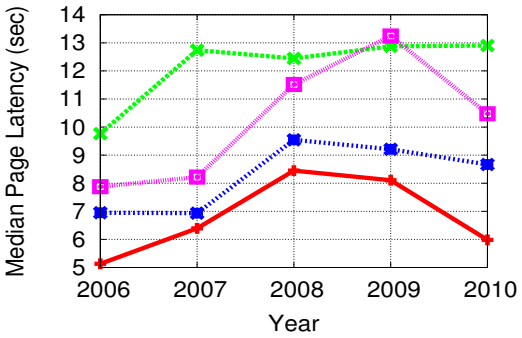


(a) % pages with Google Analytics beacon

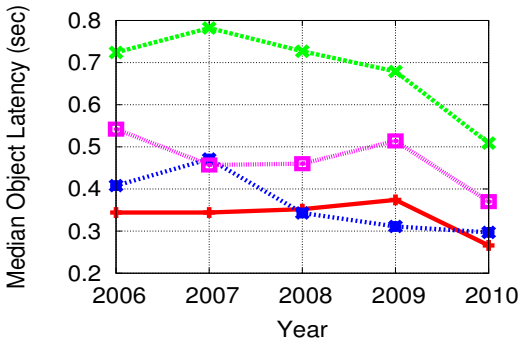
(b) Median # of objects per page



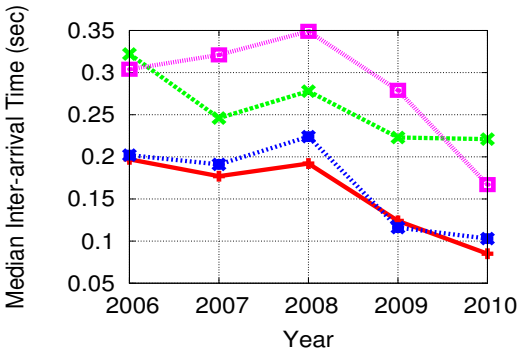
(c) Median page size



(d) Median page loading latency



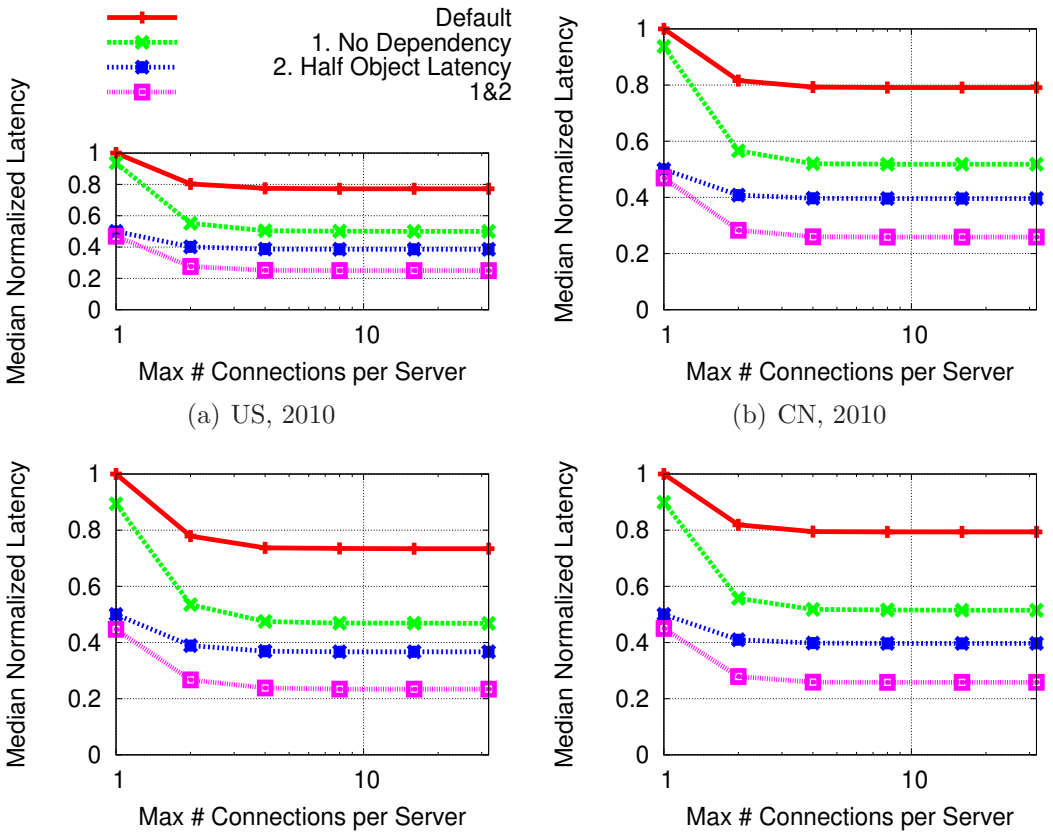
(e) Median object latency



(f) Median object inter-arrival time

Figure 2.13: Initial page characteristics: Pages have become 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.

While the page loading latency also sees a general increase in Figure 2.13 (d) until 2008, we observe a 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 decreases to 5.98 seconds in 2010. This decrease is likely due to the increased



(a) US, 2010 (b) CN, 2010

(c) FR, 2010 (d) BR, 2010

Figure 2.14: Page loading latency simulation: 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 reduce page-load latency by 67% because of the simultaneous connections. Together, page loading latency can be reduced by up to 75%.

number of concurrent connections in Figure 2.5. Another possible explanation for this decrease is the reduced latency of fetching an object in Figure 2.13 (e), which makes the object inter-arrival rate burstier as shown in Figure 2.13 (f). Since the object size does not decrease over time, the reduction of object latency is likely related to the improved client bandwidth in Figure 2.3, 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

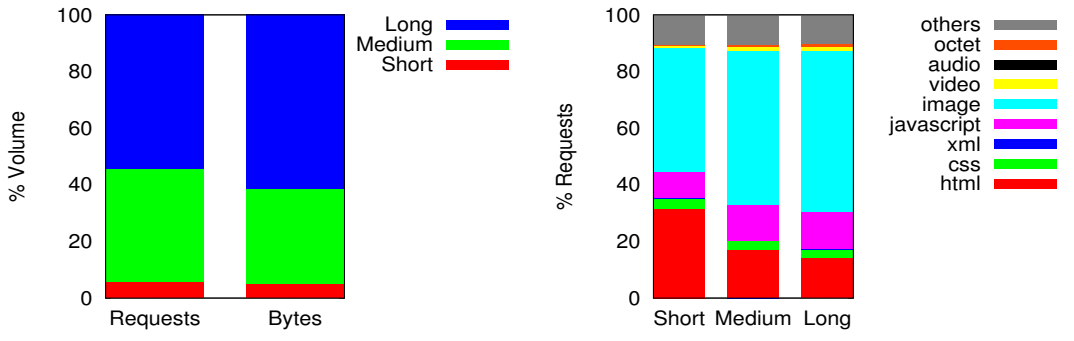
latency, and dependency among objects, we examine the impact of these factors via simulations. For simplicity, each object is fetched from a central FIFO queue, and we use the measured object latency in the access logs for the simulated object latency. The object dependency is extracted from the referer relations. We underestimate the page loading latency by ignoring network latency and browser parsing/processing time – there is no idle time. At the same time, we overestimate the latency as any dependent object cannot be fetched before its parent object is finished. [6]

Figure 2.14 presents the median simulated latency from the United States, China, and Brazil in 2010, as a function of the maximum number of concurrent connections per server. We simulate four different scenarios where 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%. But the latency converges at around 8 concurrent connections per server, increasing beyond this number of connections offers no further benefit. Second, we simulate the ideal case where there is no object dependency (`NoDep`) – all of the object URLs are known in advance, and this case 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, the page loading latency can be reduced by up to 61% due to the simultaneous connections. All together, we observe a potential reduction in page loading latency by 75%.

2.3.4 Simple Web Traffic Model

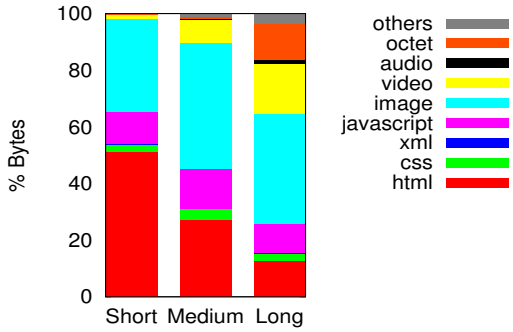
With our analysis, we are now able to derive a model of modern Web traffic. For a simple Web traffic model, we divide all of the Web pages including client-side interactions into three groups based on the total page time – short (0-25th percentile),

[6] In practice, a browser typically starts fetching embedded objects as soon as it finds their URLs.

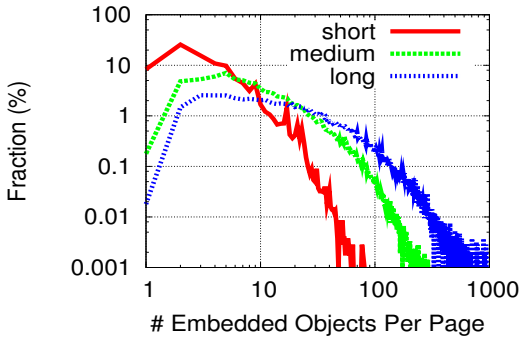


(a) % Page volume

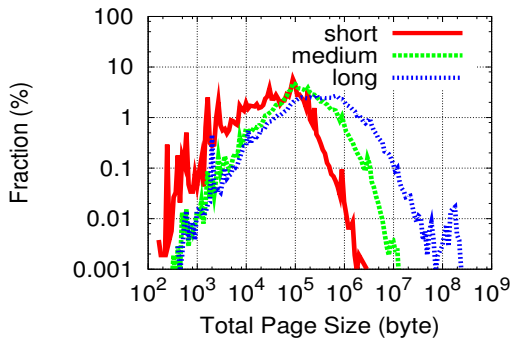
(b) Page content type by % requests



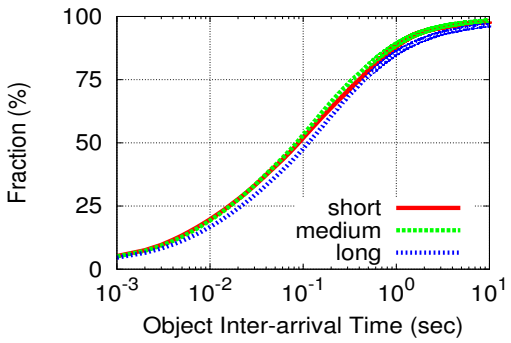
(c) Page content type by % bytes



(d) # of objects



(e) Total size



(f) Object inter-arrival time

Figure 2.15: Example Web traffic model 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.

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 2.15 presents an example traffic model from the United States in 2010.

Figure 2.15 (a) shows the volume of three different pages. In terms of requests, long pages consume about 55%, 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 2.15 (b) and (c) reveals the characteristics of the pages more clearly. Short pages are mainly HTML-oriented, such as those related to search activities. On the other hand, long pages show a higher percentage of video and octet-stream bytes than others, meaning these are mainly related to video watching activities and large file downloads. Medium pages lie in between, such as those for reading news or blogging.

In terms of the number of embedded objects, most short pages have less than 10 objects, while medium and long pages have a larger number of embedded objects, as in Figure 2.15 (d) where we show PDFs of the number of embedded objects. The median is 4, 12, and 30 for short, medium, and long pages, respectively. In particular, we observe heavy client-side interactions in long pages. Note that medium pages include Web pages that have many embedded images, however, as Web sites improve cacheability via best practices, we observe only 12 fetches for the changing content. This in part explains why page loading latency is improving despite the increase in page complexity as discussed in Section 2.3.3.

In addition, from Figure 2.15 (e) which shows PDFs of the total page sizes, we observe that the difference in medians is about 3x in size 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, Figure 2.15 (f) presents the object inter-arrival time (CDFs), and 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.

	US	CN	FR	BR
# Requests (K)	8611	12036	2129	4018
Volume (GB)	198	218	42	79

Table 2.2: Summary of captured full content data (cache-misses only)

2.4 Redundancy and Caching

The last part of our study is to analyze the redundancy in the Web traffic and the impact of caching. For this study, we consider the traditional object-based HTTP caching as well as content-based caching which is the use of caching techniques at the sub-object or packet level. In Section 2.4.1, we examine the changes in URL popularity during the five-year period with the access log data. In April 2010, we captured the full content of all traffic instead of just HTTP headers. Using this data, we directly compare the effectiveness of object-based caching and content-based caching in Section 2.4.2, and quantify the origins of redundancy in Section 2.4.3. We also calculate the actual byte hit rate with practical cache storage sizes in Section 2.4.4. Finally, we analyze the characteristics of aborted transfers, and discuss its implications on caching in Section 2.4.5. Table 2.2 shows the summary of our content data set from April, 2010. We capture cache misses only to account for simple improvements, such as using a local proxy cache.

The summary of our findings are as follows:

- We observe two interesting trends in URL popularity over time: 1) the popular URLs become more popular, but 2) the long tail of the content is also growing. However, cache hit rate is affected by both factors, and we do not observe any consistent changes in cache hit rate over time.

- We find that content-based caching yields almost 2x larger byte hit rates than object-based caching. In terms of content types, text resources such as HTML, JavaScript, XML, and CSS show higher redundancy than binary resources such

as video and image. Also, content-based caching works well regardless of content types, but the object-based caching is mainly effective for JavaScript and image only.

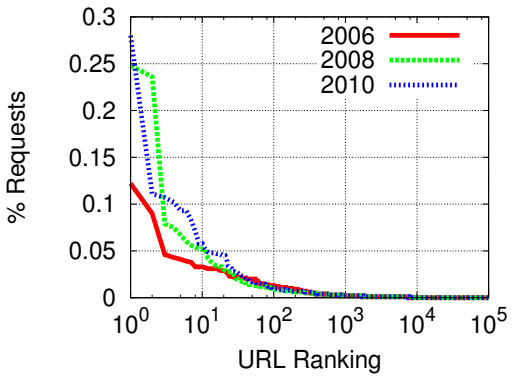
- We quantify the origins of redundancy to understand where the additional savings of content-based caching comes from. It turns out that most of the additional savings are due to partial content overlap – the redundancy across different versions of an object as well as redundancy across different objects.

- Multi-resolution chunking (MRC) [43] with large cache storage provides close to the ideal byte hit rate as it increases the working set size and reduces the metadata overhead. On the other hand, the basic content-based caching has the same working set size as object-based caching, so increasing cache beyond that working set size does not provide further benefits.

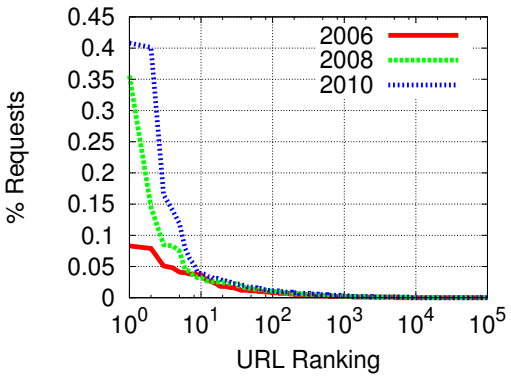
- A small number of requests (1.8-3.1%), mostly video, are aborted by users before they are fully downloaded. However, the volume is very significant (69.9-88.8%) and negatively impacts the performance of the object-based caching proxy.

2.4.1 URL Popularity

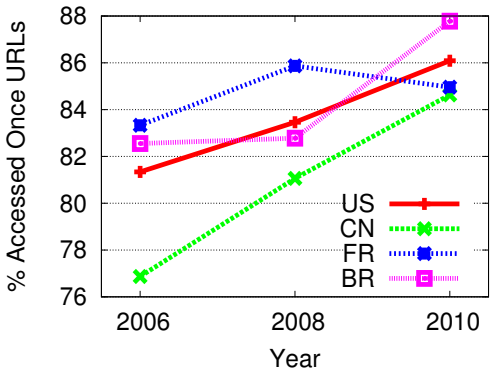
We first investigate the underlying changes in URL popularity during the five-year period with our access log data set, which directly influences the caching effectiveness. We find two interesting trends. First, we observe that the popular URLs are getting more popular as seen in Figure 2.16 (a) and (b) where we present the request percentage of the top 100,000 URLs in the United States and China. The fraction of the most popular URL sees an increase from 0.08-0.12% in 2006 to 0.28-0.41% in 2010, which 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`, however it is uncacheable. At the same time, we also find that the percentage of URLs that are



(a) US: Top 100K URLs by % requests



(b) CN: Top 100K URLs by % requests



(c) % accessed once URLs (tail)

Figure 2.16: URL popularity: The popular URLs grow, but the long tail of the content is also growing.

accessed only once is consistently increasing as in Figure 2.16 (c). We see its increase from 76.87-83.33% in 2006 to 84.63-87.78% in 2010. Overall, those URLs account for a significant amount of traffic – 29.97-48.85% of total requests and 27.27-63.94% 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 partly because the two effects cancel each other out, and also because cache hit rate is determined by other factors such as user population. In order to get an upper bound on the object-based cache hit rate with our access log data set, we assume every object is cacheable, and two objects

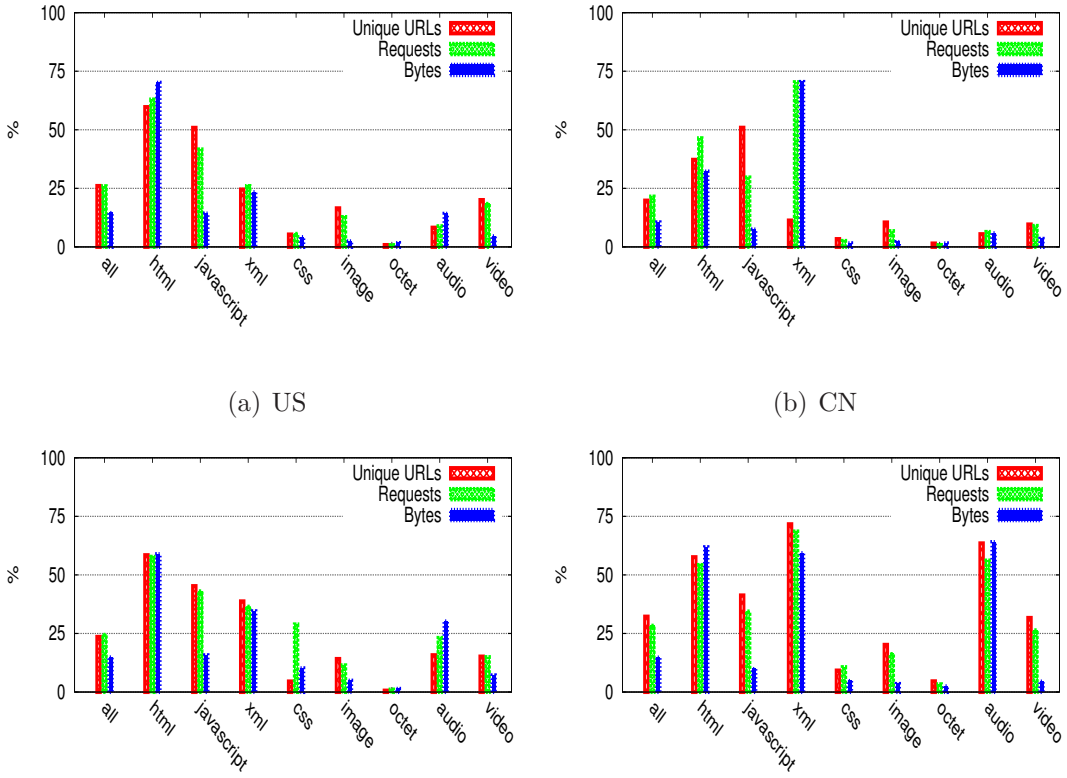
are identical (cache hit) once their URLs and content lengths match. The estimated cache hit rate we observe ranges from 35.64% to 54.48%, and the byte hit rate ranges from 15.06% to 49.27%. The byte hit rate is smaller than the cache hit rate because cache hits are biased towards relatively smaller objects. In fact, we observe 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.

2.4.2 Caching Effectiveness

In this section, we first investigate HTTP cacheability of the traffic, and calculate the ideal 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.

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's fingerprinting [80] – 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 bytes assuming the uniform distribution of content value. 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 systems sees the same chunk, it can pass only the small size of chunk names 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 the object-based caching. Furthermore,



(a) US

(b) CN

(c) FR

(d) BR

Figure 2.17: Uncacheable objects by content types: Overall, 19.80-32.20% of unique URLs are uncacheable, which accounts for 21.50-28.32% of total requests and 10.48-14.81% of total bytes. HTML and JavaScript are dynamically generated and thus less cacheable.

content-based caching is protocol independent and effective for uncacheable content as well. We discuss many related work on content-based caching in Section 2.5.

HTTP Cacheability We first 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 2.17 shows the percentage of uncacheable objects in terms of the number of unique URLs, total requests, and total bytes. Overall, 19.80-32.20% of unique URLs are uncacheable, which accounts for 21.50-28.32% of total requests and 10.48-14.81% of total bytes.

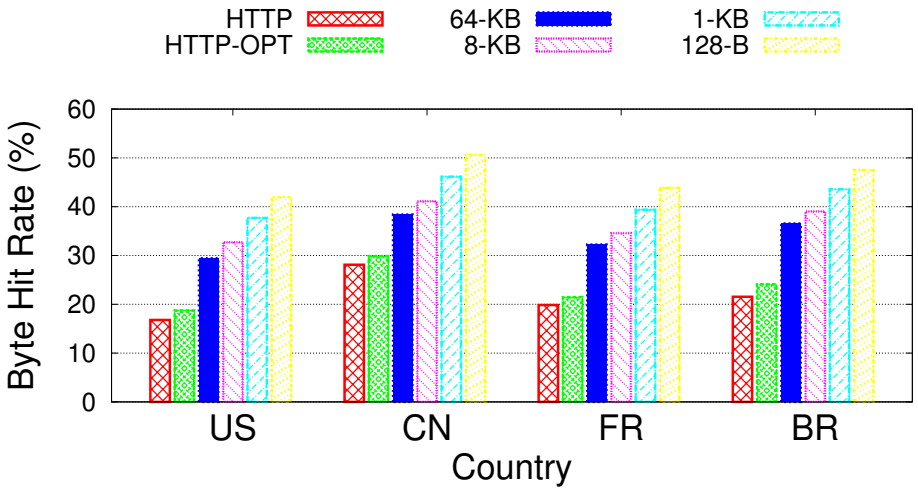


Figure 2.18: Ideal cache hit rate with infinite cache storage: Content-based caching with 128-bytes chunks achieves almost 2x larger byte hit rate than the object-based HTTP caching.

Among different content types, HTML and JavaScript are less cacheable than other content types, implying that they are dynamically generated and updated frequently.

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) [87] feeds – the two RSS URLs are responsible for 90.85% of total uncacheable bytes and 64.84% of total uncacheable requests. Second, Brazil shows a higher fraction of uncacheable XML and audio traffic than other countries. This is due to the popular use of real-time updates of sports games and live streaming of audio.

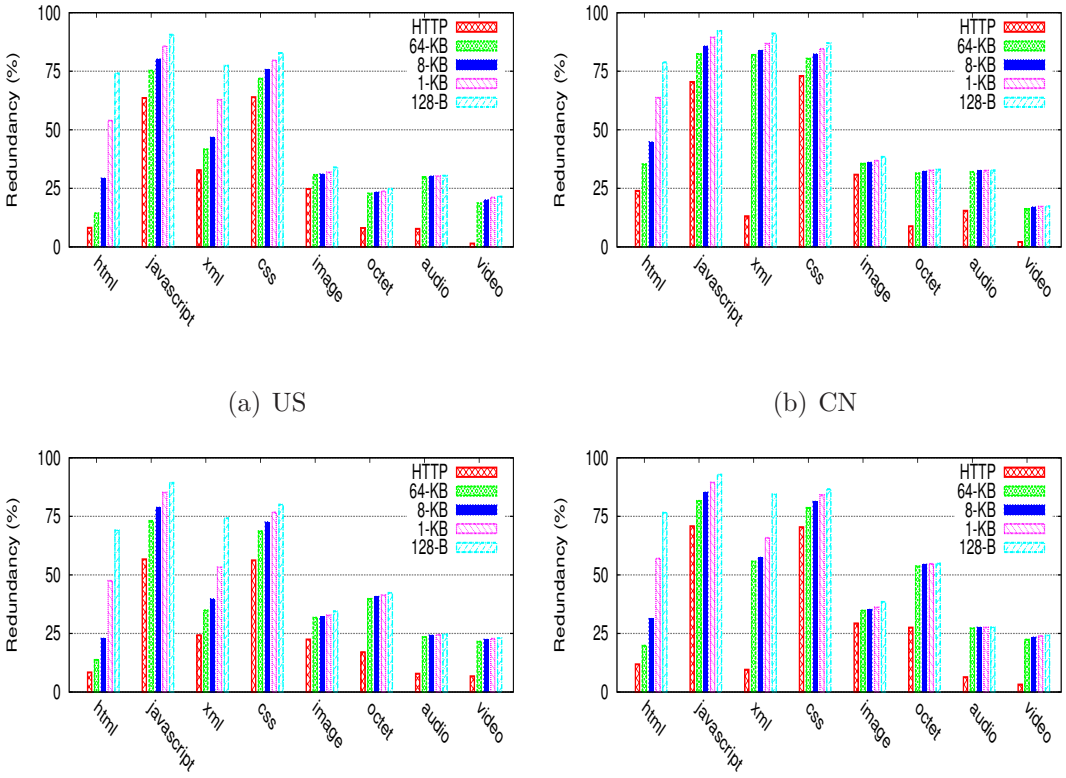
Ideal Byte Hit Rate We calculate the ideal bandwidth savings achievable with infinite cache storage by the traditional object-level HTTP caching and content-based caching. For the object-level caching, we decide if an object is cacheable with the response header. If cacheable, we check if the URLs and content lengths match as in Section 2.4.1. We also calculate a slightly optimistic behavior of the 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, 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 2.18, we observe that content-based caching outperforms the object-based caching with any chunk size. The cache hit rate of object-level caching ranges from 27.02-37.14% (not shown in the figure), but the actual byte hit rate is only 16.79-28.11%, which is similar to what previous studies reported [13, 36, 45, 60, 106]. 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 29.43-38.39% with 64 KB chunks, and the highest byte hit rate is 41.98-50.62% with 128 byte chunks, 1.8-2.5x larger than the 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 byte chunks can detect redundancy at the sentence-level, but with 64 KB chunks, redundancy can be eliminated 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 2.19 shows the ideal redundancy by content type. In particular, JavaScript shows over 90% redundancy with the smallest chunk size of 128 bytes. On the other hand, video exhibits a much lower redundancy of 20%, illustrating the impact of long-tailed popularity in video content. Object-based caching performs very poorly and its redundancy of XML in Brazil is 8x smaller than the redundancy with 128 byte chunks.

We also find that content-based caching works well regardless of the content types, while object-based caching is mainly effective for JavaScript and image traffic only.



(a) US

(b) CN

(c) FR

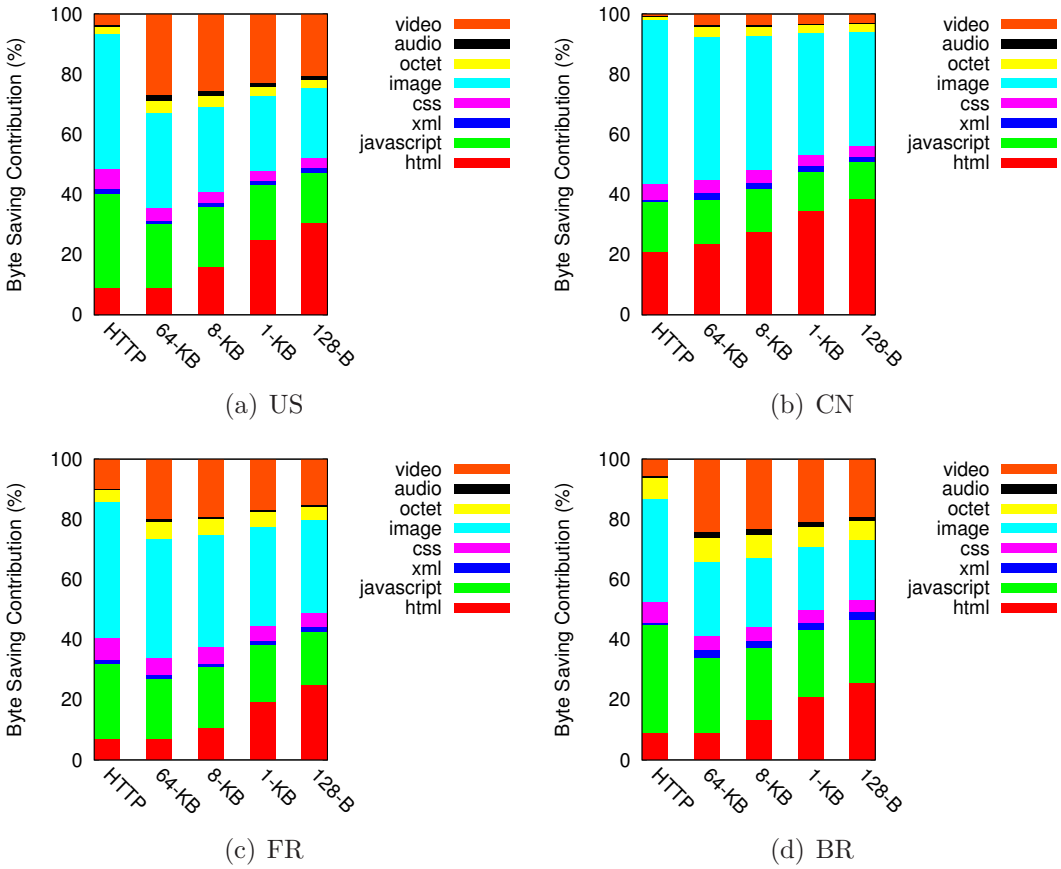
(d) BR

Figure 2.19: Ideal redundancy by content types: Text resources have higher redundancy than binary.

Figure 2.20 depicts the contribution of byte savings, showing which caching scheme works best for which content type. In the object-based caching, the contribution of JavaScript and image traffic 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, the distribution of the contribution by content type is less skewed in content-based caching.

2.4.3 Origins of Redundancy

In order to understand how content-based caching approaches provide a more than 2x larger byte hit rate than the object-based caching approach, we quantify the



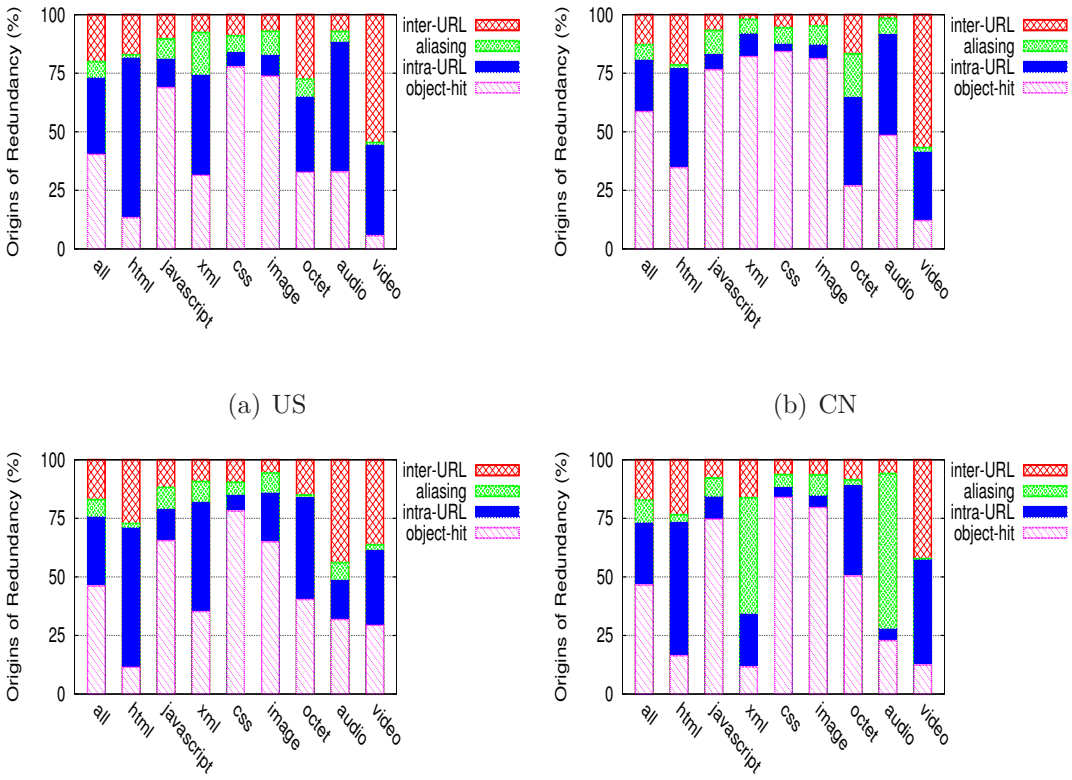
(a) US (b) CN

(c) FR (d) BR

Figure 2.20: Byte saving contribution by content types: Content-based caching is effective for any content type, but the object-based caching works well only for JavaScript and image.

contribution of redundancy from different sources in Figure 2.21. We use the average chunk size of 128-bytes for this analysis.

Overall, we observe that about 40.33-58.64% of the total redundancy is due to the identical objects with the same URLs (`object-hit`), which is essentially the upper bound of the object-based caching approach's byte hit rate. The remaining half of the total redundancy is purely from the content-based caching approach, and we further break it into the following three sources. First, there exists redundancy across the content changes of an object (`intra-URL`), accounting for about 21.83-32.55% of the total redundancy. Second, some objects with different URLs actually have the identical content [47] (`aliasing`), taking about 6.72-9.80% of the total redundancy.



(a) US

(b) CN

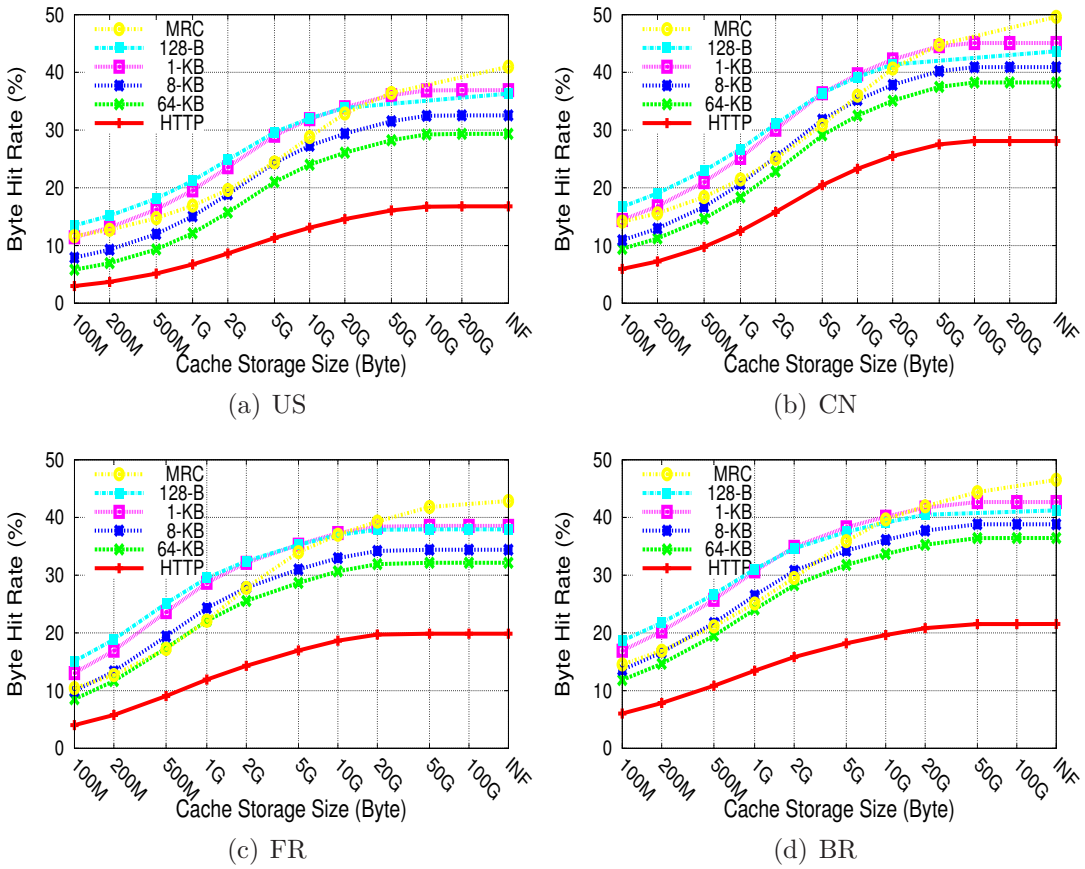
(c) FR

(d) BR

Figure 2.21: 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.

Finally, the rest is due to the redundancy across different objects that have different URLs and non-identical content (`inter-URL`), accounting for about 12.81-20.05% of the total redundancy. This analysis result implies that most of the additional savings from the content-based caching approaches come from its ability 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. This implies that such types are frequently updated but their content changes slowly. Also, generally aliasing accounts for a small amount of the total redundancy, but we observe a significant



(a) US (b) CN

(c) FR (d) BR

Figure 2.22: Cache size vs. byte hit rate: A large cache with MRC provides 2x the byte hit rate than the object-based caching.

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 the 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 2.4.5.

2.4.4 Cache Storage Size

We simulate cache behavior with different cache storage sizes to determine the required cache storage size for achieving a close to the ideal byte hit rate. This time, we 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 investigating more sophisticated policies for future work [74].

In addition to the object-based and content-based caching, we also simulate the multi-resolution chunking (MRC) that has been proposed recently, which simultaneously exploits multiple chunk sizes [43]. In detail, MRC always favors large chunks over small ones and uses small chunks only when large chunks are a cache-miss. It also caches all of the different chunk sizes for the same content redundantly for future reference. This way, MRC minimizes the metadata overhead, disk accesses, and memory pressure at the cost of more disk space.

Figure 2.22 shows our simulation results in the United States, China, France, and Brazil. [7] First, content-based caching always outperforms the 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. Second, the saturation point of cache size is similar across the different caching approaches except for MRC. For instance, 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 sized chunks start to pay off.

[7] A few missing data points are 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 as observed in a previous study [106], 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 the developing regions where bandwidth is much more expensive than disk storage [42]. 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 a close to the ideal byte hit rate with MRC. It is roughly four times the total traffic size because MRC uses four different chunk sizes in 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 we discuss in Chapter 3.

2.4.5 Aborted Transfers

In Table 2.3, 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 abortion is 12.4-30.8%. If the content was fully downloaded, it would contribute to 69.9-88.8% of the entire traffic. The investigation of the content type distribution of these transfers in Table 2.4 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. We also observe users canceling file downloads.

	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 2.3: Aborted transfers

Ranking	US	CN
1	video/x-flv (67.1%)	app/octet-stream (39.8%)
2	app/octet-stream (12.2%)	video/x-flv (38.7%)
3	video/flv (5.5%)	app/zip (3.0%)
4	video/mp4 (2.8%)	text/plain (2.6%)
5	text/plain (2.5%)	app/x-gzip (2.6%)
6	flv-app/octet-stream (2.0%)	app/pdf (2.1%)
7	video/x-msvideo (1.9%)	image/jpeg (1.5%)
8	app/x-shockwave-flash (0.8%)	flv-app/octet-stream (1.4%)
9	audio/mpeg (0.6%)	video/mp4 (1.3%)
10	video/x-ms-wmv (0.6%)	app/x-msdos-program (0.9%)
Ranking	FR	BR
1	video/x-flv (67.1%)	app/octet-stream (39.8%)
2	app/octet-stream (12.2%)	video/x-flv (38.7%)
3	video/flv (5.5%)	app/zip (3.0%)
4	video/mp4 (2.8%)	text/plain (2.6%)
5	text/plain (2.5%)	app/x-gzip (2.6%)
6	flv-app/octet-stream (2.0%)	app/pdf (2.1%)
7	video/x-msvideo (1.9%)	image/jpeg (1.5%)
8	app/x-shockwave-flash (0.8%)	flv-app/octet-stream (1.4%)
9	audio/mpeg (0.6%)	video/mp4 (1.3%)
10	video/x-ms-wmv (0.6%)	app/x-msdos-program (0.9%)

Table 2.4: Top ten content types for aborted transfers by bytes: Mostly video

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 not cache them, but this just wastes the bandwidth and reduces cache hit rate. The second option is to fully download and cache them (for cacheable objects only), but this consumes significant bandwidth for downloading objects that may 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 [95]. But this requires parameter tunings and would not work well for various objects as the number of remaining bytes varies greatly and can be very large, especially for video. 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.

On the other hand, content-based caching could cache data from only the downloaded content without any configuration, thus any data received over network, even uncacheable content, is useful. As evidence of this, in Figure 2.19 the byte hit rate of video traffic is much higher with content-based caching than with object-based caching.

2.5 Related Work

Our work is similar to previous work in a number of ways. Here, we group these related works into three areas.

Tracking Traffic Changes There is a great deal of previous work analyzing traffic changes. For example, Akamai analyzes data gathered from its global server network and reports the Internet penetration rate and connection speeds for each country [11]. Also, ipoque examines traffic from its customer ISPs and universities [44] and finds 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 [28, 49, 53].

While all of these previous studies primarily focus on the analysis 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 [14,38], but their analyses on the data set from specific organizations, such

as universities or research institutes, are inherently limited. Instead, our large-scale data set, spanning a multi-year period, covers a world-wide user population.

Web Traffic Characterization The widely used Web traffic model was first proposed by Mah, in which he introduces the idle time based page detection algorithm [52]. Since then, this model has been widely adopted by many researchers for characterizing Web traffic [10, 93]. Later, Choi and Limb developed a method that simply regards every HTML object as a Web page [18]. More recently, several studies have investigated a small number of popular Ajax Web applications, such as maps and Web mails, and streaming services [15, 89].

However, all of the previous studies have limitations in that they either assume 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.

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 cacheable objects such as static text and image files – the typical cache hit rates reported in the previous work range from 35% to 50% [13, 36, 60, 106]. The byte hit rate is even worse as cache hits are biased towards smaller popular objects. A recent study reports the byte hit rate of 20% while the cache hit rate is 43% in Zambia [45]. More advanced object-based caching techniques include delta-encoding that reduces traffic for object updates [59], and duplicate transfer detection (DTD) that avoids downloading of aliased objects [60].

Spring and Wetherall extend the object-based caching into sub-packet granularity and develop a protocol independent content-based caching technique [94]. Since

then, it has been widely adapted in many applications – network file systems [7, 64], WAN acceleration [43, 84], Web caching [70, 82], and storage systems [24]. Recently, Anand *et al.* analyzed the 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 the two with real Web traffic. Our analysis result shows that content-based caching achieves up to 42-51% byte hit rates, almost twice of that of object-based caching. Furthermore, we evaluate the effectiveness of multi-resolution chunking [43], and find that 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%). This is partly because we assume a large disk-based cache while they used in-memory cache only.

2.6 Summary

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 the pages become bigger in terms of both the number of objects and size, the page loading latency sees a decrease 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, while also

being effective with aborted transfers. Most of the additional savings of content-based caching are due to the partial content overlaps.

Chapter 3

Wide-area Network Acceleration for the Developing World

While low-cost laptops may soon improve computer access for the developing world, their widespread deployment will increase the demands on local networking infrastructure. Locally caching static Web content can alleviate some of this demand, but this approach has limits on its effectiveness, especially in smaller environments.

We propose to augment these caches with integrated wide area network (WAN) accelerators that have been specifically designed to operate in developing-world environments. WAN accelerators are deployed near edge routers, and work by compressing redundant traffic destined to locations with other WAN accelerators. To compress traffic, the accelerators break the data stream into smaller chunks, store these chunks at each accelerator, and then replace future instances of this data with reference to the cached chunks. By passing references to the chunks rather than the full data, the accelerator compresses the data stream.

Current WAN accelerators are not well-suited for the developing world. While they typically require server-class machines with a set of fast disks and a large pool of dedicated memory, the average school targeted by the One Laptop Per Child (OLPC)

project will have 100 laptops in the price range of US $100-$200 each, for a total cost of $10K-$20K [67]. Requiring special server-class hardware for WAN acceleration alone could increase deployment cost. Other options would be to share the machine with other services (e.g, mail servers, Web servers, and proxies) or to use cheap, laptop-class hardware, both of which would reduce the RAM and disk available to the WAN accelerator. In addition, existing designs cannot exploit the mesh network environments being deployed in the developing world, limiting their potential utility.

We have developed a new WAN accelerator, Wanax, that is designed to meet these challenges in the developing world. Our technical contributions are the followings: (1) a novel multi-resolution chunking (MRC) technique, which provides high compression rates and high disk performance across workloads while having a small memory footprint; (2) an intelligent load shedding technique that exploits MRC to maximize effective bandwidth by adjusting disk and WAN usage as appropriate; and (3) a mesh peering protocol that exploits higher-speed local peers when possible, instead of fetching only over slow WAN links. The combination of these design techniques makes it possible to achieve high effective bandwidth even with resource-limited shared machines.

The rest of this chapter is organized as follows: Section 3.1 provides background on WAN accelerators and new challenges in the developing world. Section 3.2 describes the design of Wanax, and we show the trace-based simulation analysis in Section 3.3. In Section 3.4, we detail the prototype implementation, and Section 3.5 presents the experimental results. Finally, we discuss related work in Section 3.6, and conclude in Section 3.7.

3.1 Background and Motivation

Our goal is to improve Internet access in the developing world using WAN accelerators designed to use low-end hardware. We primarily focus on increasing the *effective bandwidth* (or throughput) of the expensive, low-bandwidth WAN link in the region. We first provide a brief introduction to WAN accelerators, and then discuss the specific problems.

3.1.1 WAN Accelerators

Content Fingerprinting Content fingerprinting (CF) forms the basis for WAN acceleration, since it provides a position-independent and history-independent technique for breaking a stream of data into smaller pieces, or chunks, based only on their content.

While early systems used Manber's anchor technique to determine chunk boundaries [55], Rabin's fingerprinting technique is now widely used for its efficiency and flexibility [80]. It continuously generates integer values, or fingerprints, over a sliding window (e.g., 48 bytes) of a byte stream. When a fingerprint matches a specified global constant K, that region constitutes a chunk boundary. The average chunk size can be controlled with a parameter n, that defines how many low-order bits of K are used to determine chunk boundaries. In the average case, the expected chunk size is 2^n bytes. To prevent chunks from being too large or too small, minimum and maximum chunk sizes can be specified as well. Since Rabin fingerprinting determines chunk boundaries by content, rather than offset, localized changes in the data stream only affect chunks that are near the changes.

Once a stream has been chunked, the WAN accelerator can cache the chunks and pass references to previously cached chunks, regardless of their origin. As a result,

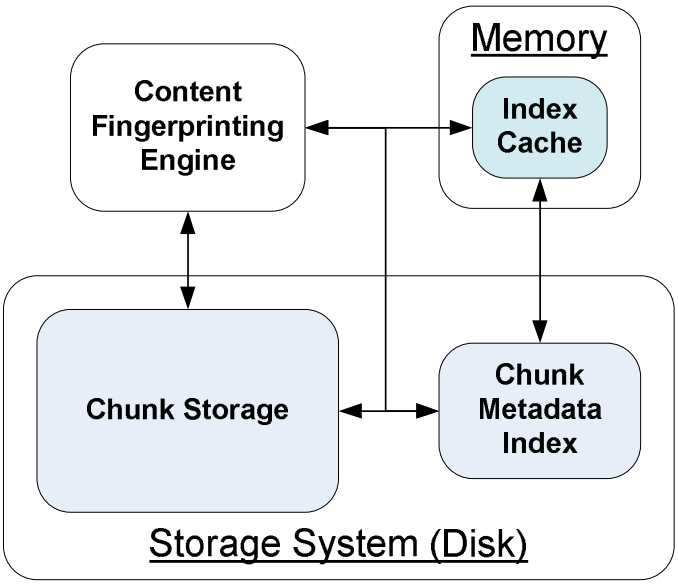


Figure 3.1: WAN Accelerator Architecture

WAN accelerators can compress within a stream, across streams, and even across files and protocols.

Performance Trade-offs Figure 3.1 depicts the general architecture of modern WAN accelerators. Chunk data is stored on disk due to cost and capacity, but an index of chunk metadata is partially or completely kept in memory to avoid disk accesses. Memory also serves as a cache for chunk data, to reduce disk access for commonly-used content.

The performance of WAN accelerators is mainly determined by three factors - (1) *compression rate*, (2) *disk performance*, and (3) *memory pressure*. Compression rate refers to the fraction of the original data that actually gets sent, and reflects network bandwidth savings by receiver-side caching. Disk performance determines the cached chunk access time (seek time) while memory pressure affects the efficiency of the chunk index and in-memory caching. These three factors affect the total latency, which is the time to reconstruct and deliver the original data. Delivering high effective bandwidth

requires reducing the total latency – having *high compression*, *low disk seeks*, and *low memory pressure* simultaneously.

Chunk size directly impacts all three factors, and consequently the effective bandwidth as well. Small chunks can lead to better compression if changes are fine-grained, such as a word being changed in a paragraph. Only the chunk containing the word is modified, and the rest of the paragraph can be compressed. However, for the same storage size, smaller chunks create more total chunks, increasing the metadata index size, and increasing the memory pressure and disk seeks. Large chunks yield fewer chunks in total, reducing memory pressure from indexing and providing better disk usage since each read can provide more data. Large chunks, however, can miss fine-grained changes, leading to lower compression. No chunk size is standard in systems that use content fingerprinting – for example, VBWC [82] uses a 2KB chunk size, LBFS [64] uses 8KB, and Shark [7] uses 16KB.

3.1.2 Developing World Challenges

Our target environment, schools in the developing world, is very different from enterprise branch offices, the typical candidate for WAN accelerators.

Limited RAM Due to cost, schools want a *shared machine* or a cheap laptop with limited RAM running the WAN accelerator and other services, instead of using a dedicated server appliance. Also, school children may want to access any content on the Internet, rather than just a smaller set of work-related documents in the enterprise environment. This *larger working set* requires more disk storage, more chunks, and more metadata entries, increasing memory pressure.

Poor Disk Performance While disk capacity is cheap and large (1TB SATA per $100), disk seek performance is still limited and is often the bottleneck. Modern desktop drives typically perform roughly 100 seeks/second, but cheaper laptop/external

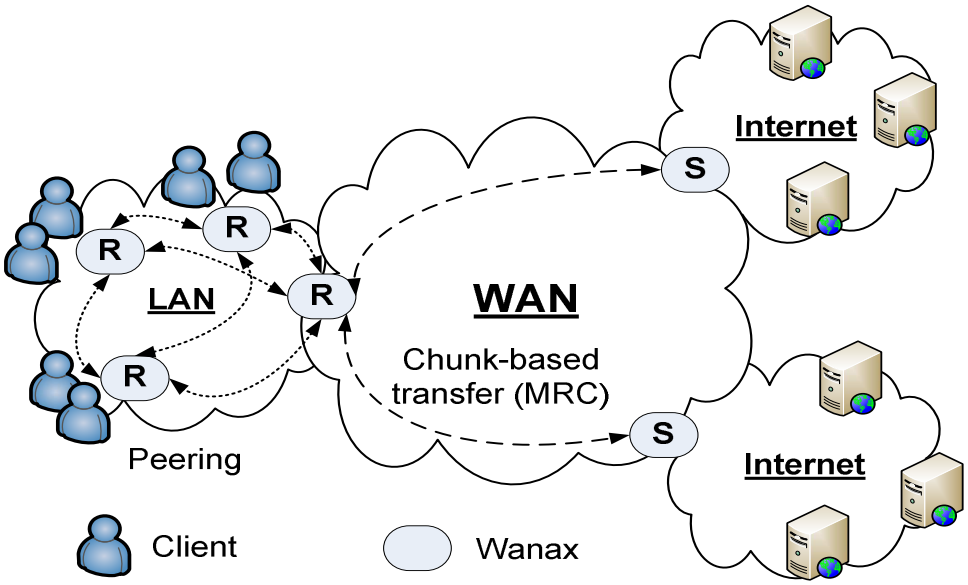


Figure 3.2: Wanax System Overview

drives we may expect in the developing world are even slower, and are much slower than the high-RPM SCSI disks commercial WAN accelerators use. Also, the larger working set and other services sharing the disks further increase the disk load.

Low Compression Rate To handle poor disk performance in the developing world, one choice is to use large chunks to reduce the number of disk accesses, but this reduces the compression rate, limiting bandwidth gains.

Mesh Topology Enterprise branch offices typically communicate with a central office in a star topology, whereas many schools in a local region may prefer to get content from each other over cheaper local links rather than over the WAN link. Current WAN accelerators are not designed to exploit this opportunity.

3.2 Wanax Design

Motivated by the challenges in the developing world, we design Wanax around four goals - (1) maximize compression, (2) minimize disk seeks, (3) minimize memory pressure, and (4) exploit local resources.

Wanax works by compressing redundant traffic between a pair of servers – one near the clients, called a R-Wanax, and one closer to the content, called an S-Wanax. For developing regions, the S-Wanax is likely to be placed where bandwidth is cheaper. For example, in Africa, where Internet connectivity is often backhauled to Europe via slow and expensive satellite, the S-Wanax may reside in Europe.

Since we expect most Wanax usage will be Web-related, Wanax operates on TCP streams rather than IP packets. Also, buffering TCP flows yields larger regions for content fingerprinting than packet boundaries, providing an opportunity for high compression rate. The remote Wanax divides the incoming TCP stream into chunks and sends chunk identifiers (such as SHA-1 hashes) to the local Wanax. If the local Wanax has the chunks cached, the data is reassembled and delivered to the client. Any chunks that are not cached can be fetched from the remote Wanax or other nearby peer. Figure 3.2 shows the overall system architecture. Each machine is capable of acting as both S-Wanax and R-Wanax, based on the direction of communication.

3.2.1 Basic Protocol

Wanax uses three kinds of communication channels between the accelerators – control, data, and monitoring channels. The control channel is used for connection management and chunk name exchange. The data channels are used to request and deliver uncached chunks, so it is stateless and implemented as a simple request-reply protocol. Finally, the monitoring channel is used for checking the liveness and load levels

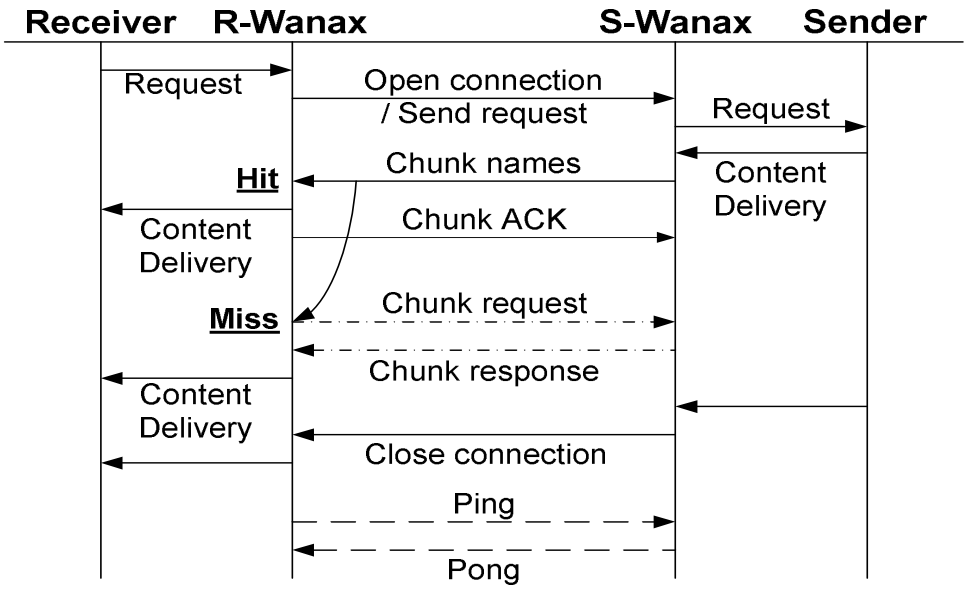


Figure 3.3: Basic Protocol

of the peers using a simple heartbeat protocol. Figure 3.3 shows typical data transfer between two Wanax gateways.

Control Channel When client A initiates a TCP connection to client B in the WAN, that connection is transparently intercepted by the Wanax gateway accelerator [8], R-Wanax. R-Wanax selects S-Wanax which is network topologically closer to B, and sends it an *open connection* message with the IP and the port number of B. S-Wanax then opens a TCP connection to B and a logical end-to-end user connection between A and B is established.

When the client B sends data back to S-Wanax, S-Wanax generates chunk names from the data and sends them to R-Wanax in a *chunk name* message. Each chunk name message contains a sequence number so that R-Wanax can reconstruct the original content in the right order. After R-Wanax reconstructs and delivers the chunk data to the original client, it sends a *chunk acknowledgment* (ACK) message to S-Wanax. S-Wanax can then safely discard the delivered chunks from its memory, and proceed with sending more chunk names.

[8]Non-cacheable protocols (*e.g.*, SSH, HTTPS) are bypassed.

When the sender or receiver closes the connection, the corresponding Wanax sends a *close connection* message to other gateway and the connections between the gateways and the clients are closed once all the data is delivered. The control channel, however, remains connected. All control messages carry flow identifiers, so one control channel can be multiplexed for many data flows. Control messages can be batched for efficiency.

Data and Monitoring Channels The data channel uses *chunk request* and *chunk response* messages to deliver the actual chunk content in case of a cache miss at R-Wanax. We also have the *chunk peek* message for querying if a given chunk is cached, which is used in our load shedding system.

Each Wanax accelerator monitors the status of its peers by exchanging heartbeats on the monitoring channel. The heartbeat response carries the load level of disk and network I/Os of the peer so that we can balance the request load among peers. We present the details of our load shedding scheme shortly in Section 3.2.4.

3.2.2 Multi-Resolution Chunking

MRC combines the advantages of both large and small chunks by allowing multiple chunk sizes to co-exist in the system. Wanax uses MRC to achieve (1) high compression rate, (2) low disk seeks, and (3) low memory pressure. When content overlap is high, Wanax can use larger chunks to reduce disk seeks and memory pressure. However, when larger chunks miss compression opportunities, Wanax uses smaller chunk sizes to achieve higher compression. In contrast, existing WAN accelerators typically use a fixed chunk size, which we term *single-resolution chunking*, or SRC.

Generating Chunks Generating multiple chunk sizes requires careful processing, not only for efficiency, but also to ensure that chunk boundaries are aligned. A naive approach to generating chunks can yield unaligned chunk boundaries, as shown

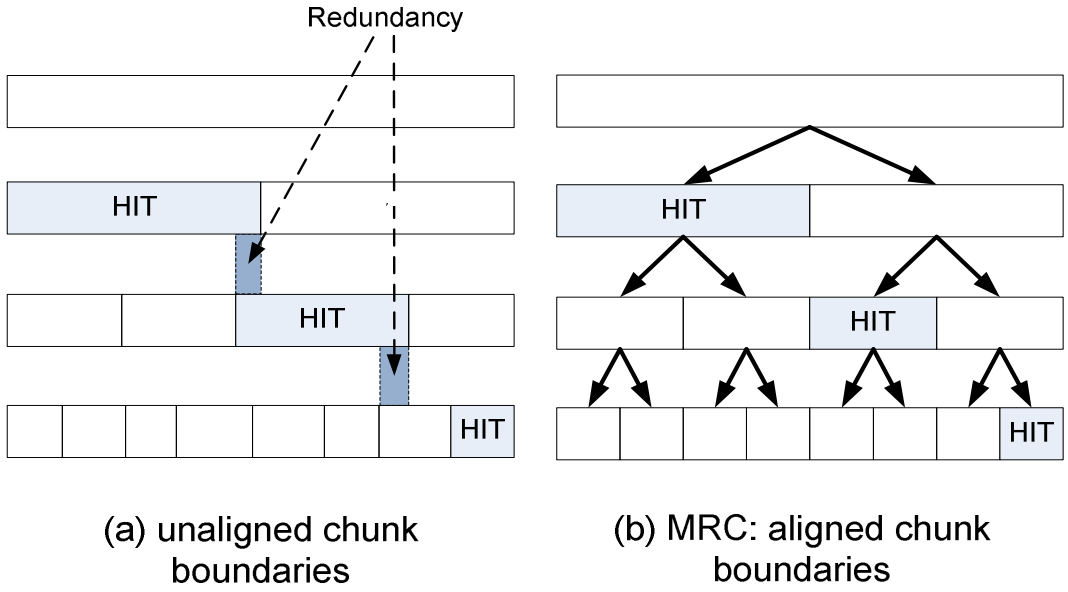


Figure 3.4: Multi-Resolution Chunking

in Figure 3.4(a). Here, the fingerprinting algorithm was run multiple times with multiple sizes. However, due to different boundary-detection mechanisms, chunk size limits, or other issues, the boundaries for larger chunks are not aligned with those of smaller chunks. As a result, when fetching chunks to reconstruct data, some areas of chunks overlap, while some chunks only partly overlap, causing wasted bandwidth when a partially-hit chunk must be fetched to satisfy a smaller missing range.

Instead, we perform a single-pass fingerprinting step, in which all of the smallest boundaries are detected, and then larger chunks are generated by matching different numbers of bits of the same boundary detection constraint. This process produces the *MRC tree* shown in Figure 3.4(b), where the largest chunk is the root, and all smaller chunks share boundaries with some of their leaf chunks. Performing this process using one fingerprinting pass not only produces a cleaner chunk alignment, but also requires less CPU.

Storing Chunks All chunks generated by the MRC process are stored to disk, even though the smaller chunks contain the same data as their parent. The rationale

Scheme	Compression Rate	Disk I/O	Memory Pressure	Index Update
SRC-Small	High	High	High	Simple
SRC-Large	Low	Low	Low	Simple
MRC-Small	High	High	High	Complex
MRC-Large	High	Low	High	Complex
MRC	High	Low	Low	Simple

Table 3.1: Comparison of Chunking Schemes

behind this decision is based on the observation that disk space is cheap, and having all chunks be fully independent simplifies the metadata [9] indexing process, reducing memory pressure in the system, also minimizing disk seeks as well. For example, when reading a chunk content from disk, MRC requires only *one* index entry access, and only *one* disk seek.

Two other options would be to reconstruct large chunks from smaller chunks, which we call *MRC-Small*, and storing the smaller chunks as offsets into the root chunk, which we call *MRC-Large*.

While both MRC-Small and MRC-Large can reduce disk space consumption by saving only unique data, they suffer from more disk seeks and higher memory pressure. To reconstruct a larger chunk, MRC-Small needs to fetch all the smaller chunks sharing the content, which can significantly increase disk access. The metadata for each small chunk is accessed in this process and loaded in memory, increasing memory pressure compared to standard MRC with only one chunk index entry. MRC-Large avoids multiple disk seeks but complicates chunk index management. When a chunk is evicted from disk or overwritten, all dependent chunks must also be invalidated. This requires either that each metadata entry grows to include all sub-chunk names, or that all sub-chunk metadata entries contain backpointers to their parents.

MRC avoids these problems by making all chunks independent of each other. This choice greatly simplifies the design at the cost of more disk space consumption. In

[9] chunk name, disk location of chunk content, and chunk length at a minimum.

practice, however, we can store more than one month's worth of chunk data on a single 1 TB disk assuming a 1 Mbps WAN connection. Table 3.1 summarizes the trade-offs of different schemes.

Content Reconstruction When an R-Wanax receives an MRC tree (chunk names only) from an S-Wanax, it builds a *candidate list* to determine which chunks can be fetched locally, at peers, and from the S-Wanax. To get this information, it queries its local cache and peers for each chunk's status, starting from the root. Since Wanax uses the in-memory index to handle this query, it does not require extra disk access. If a chunk is a hit, R-Wanax stops querying for any children of the chunk. For misses, we find the root of the subtree containing only misses, and fetch that from S-Wanax. After reconstructing the content, Wanax stores each uncached chunk in the MRC to disk for future reference.

Chunk Name Hints Optimization Sending full MRC trees would waste bandwidth if there is a cache hit at a high level in the tree or when subtrees are all cache misses. Sending one level of the tree at a time avoids the wasted bandwidth, but increases the transmission latency with a large number of round trips. Instead, we have S-Wanax predict chunk hits or misses at R-Wanax and prune the MRC tree accordingly. We augment S-Wanax with a hint table that contains recently-seen chunk names along with timestamps. Before sending the MRC tree, S-Wanax checks all chunk names against the hint table. For any hit in the hint table, S-Wanax avoids sending the subtrees below the chunk. If it is a miss or the chunk name hint is stale, S-Wanax determines the largest subtree that is a miss and sends one chunk content for the entire subtree. This way, we eliminate any inefficiency exchanging MRC trees, further increasing effective compression rate.

Here, we assume that the S-Wanax and the R-Wanax will be synchronized over time – what an S-Wanax has in the hint table will already be in R-Wanax's cache.

We use the timestamps to invalidate old hint entries, but even if prediction is wrong, it does not affect correctness.

3.2.3 Resource Sharing via Peering

Wanax incorporates a peering mechanism to share the resources such as disks, memory, and CPU with nearby peers using cheaper/faster local connectivity. It allows Wanax to distribute the chunk fetching load among the peers and utilize multiple chunk cache stores in parallel, improving performance. In comparison, existing WAN accelerators support only point-to-point communication.

To reduce scalability problems resulting from querying peers [106], Wanax uses a variant of consistent hashing called Highest Random Weight (HRW) [100]. Regardless of node churn, HRW deterministically chooses the responsible peer for a chunk. We considered other approaches like Summary cache [29], but HRW consumes small memory at the expense of more CPU cycles, and this trade-off fits well in the developing world scenario. In comparison, periodic rebuilds of a Bloom filter would require re-scanning all chunk metadata, causing significant memory pressure and possibly disk access.

Figure 3.5 shows a scenario using Wanax peers. On receiving the *chunk name* message from S-Wanax, R-Wanax sends a *chunk request* message to its responsible peer Wanax. The message includes the missing chunk name and the address of S-Wanax from whom the name of the missing chunk originates. If the peer Wanax has the chunk, it sends the requested chunk content back to R-Wanax with a *chunk response* message. If not, the peer proxy can fetch the missing chunk from S-Wanax, deliver it to R-Wanax, and save the chunk locally for future requests. If peers are not in the same configured region/LAN and could incur separate bandwidth cost, fetching the missing chunk falls back to the R-Wanax instead of the peer. After finishing data

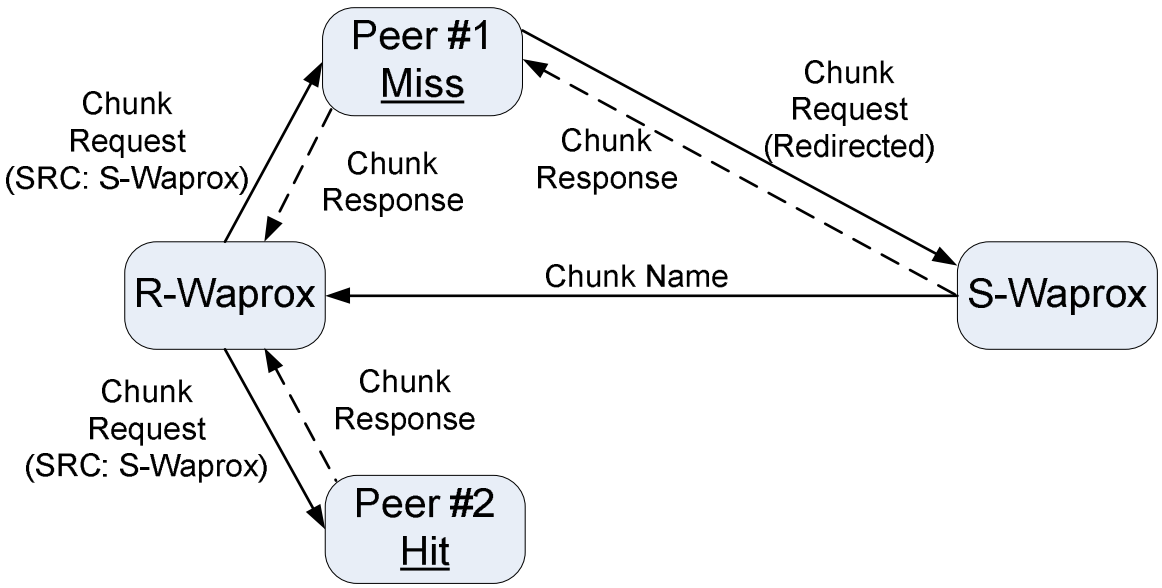


Figure 3.5: Getting Chunk Content from Peers

reconstruction, R-Wanax also distributes any uncached chunk to its corresponding peers. We introduce a *chunk put* message in the data channel for this purpose.

3.2.4 Intelligent Load Shedding

While chunk cache hits are desirable in general since they reduce bandwidth consumption, too many disk accesses may degrade the effective bandwidth by increasing the overall latency. This problem becomes even worse in the developing world where the disk performance is poor. In such cases, we can opportunistically use network bandwidth instead of queueing more requests to the disk. By using the disk for larger chunks and fetching smaller chunks over the network, we can sustain high effective bandwidth without disk overload.

We introduce intelligent load shedding (ILS), which exploits the structure of the MRC tree and dynamically schedules chunk fetches to maximize the effective bandwidth given a resource budget. The ILS algorithm is presented in Algorithm 1, and takes the link bandwidth (BW) and round-trip latency (RTT) of the R-Wanax as input. Each peer Wanax also uses the monitoring channel to send heartbeats that

Algorithm 1 Intelligent Load Shedding

Require: C: all the chunk names to be scheduled
BW, RTT: link bandwidth and RTT
Q_i: # of pending disk requests for peer i
B_i: pending network bytes to receive for peer i
S: per chunk disk latency

1: partition C with HRW
2: resolve C with *chunk peek* message in parallel
3: generate the candidate list
D_i: cache-hit chunks on peer i
N: cache-miss chunks
4: estimate each latency
$T_{D_i} = (|D_i| + Q_i) \times S$
$T_N = RTT + \{\sum_i B_i + \sum_{c \in N} length(c)\}/BW$
5: **while** $max(T_{D_i}) > T_N$ **do**
6: pick the peer k where $max(T_{D_i}) = T_{D_k}$
7: move the smallest chunk from D_k to N
8: update T_{D_k} and T_N
9: **end while**
10: return D_i and N

contain its network and disk load status in the form of the number of pending disk requests (Q_i), and the pending bytes to receive from network (B_i). We assume per-chunk disk read latency (S), or seek time is uniform for all peers for simplicity.

The first step in the ILS process is generating the candidate list. On receiving the chunk names from S-Wanax, R-Wanax runs the HRW algorithm to partition the chunk names (C) into responsible peers. Some chunk names are assigned to R-Wanax itself. Then R-Wanax checks if the chunks are cache hits by sending the *chunk peek* messages to the corresponding peers in parallel. Based on the lookup results, R-Wanax generates the candidate list (Section 3.2.2). Note that this lookup and candidate list generation process (line 2 and 3 in Algorithm 1) can be avoided by name hints from S-Wanax, which R-Wanax uses to determine the results without actual lookups.

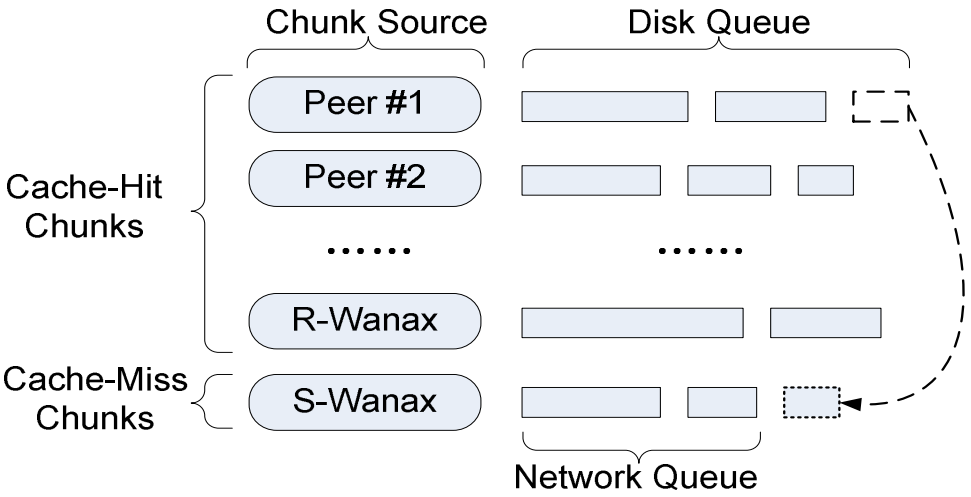


Figure 3.6: Intelligent Load Shedding: by moving smaller chunks from the disk queue to the network queue, the overall latency is further reduced.

The next step in the ILS process is estimating fetch latencies for the network and disk queues. From the candidate list, we know which chunks need to be fetched over network (*network queue*, N) and which chunks need to be fetched either from local disk or a peer (*disk queues*, D_i). Based on this information, we estimate the latency for each chunk source. For each disk queue, the estimated *disk* latency will be per-chunk disk latency (S) multiplied by the number of cache hits. For the network queue, the estimated *network* latency will be one RTT plus the total size of cache-miss chunks divided by BW. If there were pending chunks in the network or disk queues, each latency is accordingly adjusted. We assume the latency between the R-Wanax and peers is small, and do not incorporate it in our model.

The final step in ILS is balancing the expected queue latencies, but doing so in a bandwidth-sensitive manner. We decide if we need to move some cache hit chunks from a disk queue to a network queue – since fetching chunks from each source can be done in parallel, the total latency will be the maximum latency among them. If the network is expected to cause the highest latency, we stop here because no further productive scheduling is possible. When disk latency dominates, we can reduce it by fetching some chunks from the network. We choose the *smallest* chunk because

it reduces one disk seek latency while increasing the minimum network latency. We update the estimated latencies, and repeat this process until the latencies equalize, as shown in Figure 3.6. After finishing ILS, R-Wanax distributes *chunk request* messages to corresponding peers. We send the requests in the order they appear in the candidate list, in order to avoid possible head-of-line (HOL) blocking.

Note that ILS algorithm works with both MRC and SRC. However, by moving the smallest chunk from the disk queue to the network queue, MRC could further reduce the disk latency than SRC, which results in smaller overall latency. Combined with MRC's better overall disk performance and compression, it gives much higher effective bandwidth.

3.2.5 Skipping Compression

While Wanax tries to save the bandwidth consumption, there are two sources of overhead that may increase the latency – disk read latency at cache hit, and one extra RTT at cache miss. If the overhead dominates the overall latency, it is desirable not to perform compression and just forward the data.

We develop a simple model similar to DTD [60] in order to understand when it is beneficial to perform compression for reducing the overall latency. For simplicity, we assume a request-response protocol like in HTTP and ignore issues such as congestion control, slow-start, MRC, peering, and ILS.

We first consider the case without using chunk name hints. Suppose we are receiving the data of size R where the link bandwidth is BW and the round-trip time is RTT. We also assume the average chunk size is C, and the average disk access time per chunk is S. From this, we can derive T_{base}, the base delay without WAN acceleration, T_{hit}, the delay when every chunk is a cache hit, and T_{miss}, the delay when every chunk is a cache miss.

$$T_{base} = RTT + \frac{R}{BW}$$
$$T_{hit} = RTT + \frac{R}{C} \times S$$
$$T_{miss} = RTT + T_{base}$$

Assuming the average redundancy (or hit ratio) in R is H, we can express the total expected latency when we perform compression as follows:

$$T_{Wanax} = H \times T_{hit} + (1 - H) \times T_{miss}$$

The overall latency with compression will be reduced if $T_{Wanax} < T_{base}$. This condition holds when:

$$BW < \frac{H \times R}{(1 - H) \times RTT + H \times \frac{R}{C} \times S}$$
$$R > \frac{(1 - H) \times RTT}{H \times (\frac{1}{BW} - \frac{S}{C})}$$

We see that the bandwidth should be small enough and the data size should be large enough to benefit from compression. WAN acceleration provides the most benefit if the links have low bandwidth and high latency, which is what we see in developing regions.

We also consider the case when we use the chunk name hint. In this case, we can avoid an extra RTT of cache miss penalty since S-Wanax would send the content from the outset. After removing an RTT from T_{miss}, we derive the following condition:

$$BW < \frac{C}{S}$$

We arrive at the same conclusion that the bandwidth should be small enough. However, the data size or the hit ratio no longer affects the overall latency in this case. More important is that the disk latency is small enough that reading chunks from disk is faster than retrieving them over network.

Wanax automatically turns off compression when the data size is less than the pre-configured break-even data size, or if the link bandwidth is not small enough. Note that this optimization, which is performed by S-Wanax is complementary to ILS performed by R-Wanax.

3.3 Simulation Analysis

To understand the trade-offs between MRC and other schemes, we simulate their behavior under a variety of workloads, comparing bandwidth savings, disk access overheads, memory pressure, and performance.

3.3.1 Simulator

We develop a simulator that reads the packet-level traces from tcpdump [98] and simulates various scenarios using SRC and MRC. The simulator uses libnids [51] for stream reconstruction, and consists of 7,000 lines of C code. The outputs are actual and ideal bandwidth savings with and without chunk indexing metadata overhead, disk access overhead for chunk content fetching, and total memory usage. We use 20-byte SHA-1 hashes for the chunk names, and model point-to-point deployments with one S-Wanax and one R-Wanax with no peers. The simulator implements all

of the Wanax design mentioned earlier, including the chunk name hint optimization used for both SRC and MRC.

We vary the chunk size for both schemes, with SRC using chunks from 32 bytes to 64 KB, and MRC using three tree configurations, with a 64 KB root chunk with tree degrees 2, 4 and 8 each. The child chunk size is obtained by dividing the parent chunk size by the degree. For example, a degree-2 tree ($d = 2$) starts with a 64 KB root chunk and two 32 KB children chunks. Each child chunk recursively forms a subtree with the same degree until the chunk size reaches 32 bytes. A degree-4 tree has 64 bytes as leaf node size while a degree-8 tree has 128 bytes as the minimum size. If needed, we also change the height of the MRC tree of the same degree, by controlling the smallest chunk size, m.

3.3.2 Workload

We choose two types of workloads – dynamically-generated Web content and redundant large files. We focus on dynamic content because the static content is likely to be handled by a standard Web proxy, and we can further reduce bandwidth consumption on uncacheable content with Wanax. We select a number of popular news sites, fetch the front pages every five minutes, and measure the redundancy between the fetches. [10] To generate traffic close to what actual users would produce, we use Firefox 3.0 [62] to fetch the content, and we enable the browser cache to avoid re-fetching cacheable content. We collect packet-level traces for three days, yielding a 1GB trace with 102K TCP sessions and a 72% redundancy. We refer to this workload as "news sites".

The large-file workload represents long-lived connections for videos or software packages. For this, we download two different versions of the Linux kernel source tar files, 2.6.26.4 and 2.6.26.5, one at a time and gather packet-level traces as well. The

[10] CNN, Google News, NYTimes, Slashdot, Digg, Fark, Salon, Yahoo News, and Drudgereport.

	SRC	MRC-2	MRC-4	MRC-8
H-U/W-U	20	20	21	23
H-U/W-C	62	62	61	59
H-C/W-C	10	10	9	8
H-C/W-U	8	8	9	10

Table 3.2: News Sites Cacheability Breakdown (%) – as a result of browser caching, most traffic in this workload is HTTP-uncacheable (H-U). However, it still has much redundancy, making most bytes Wanax-cacheable (W-C).

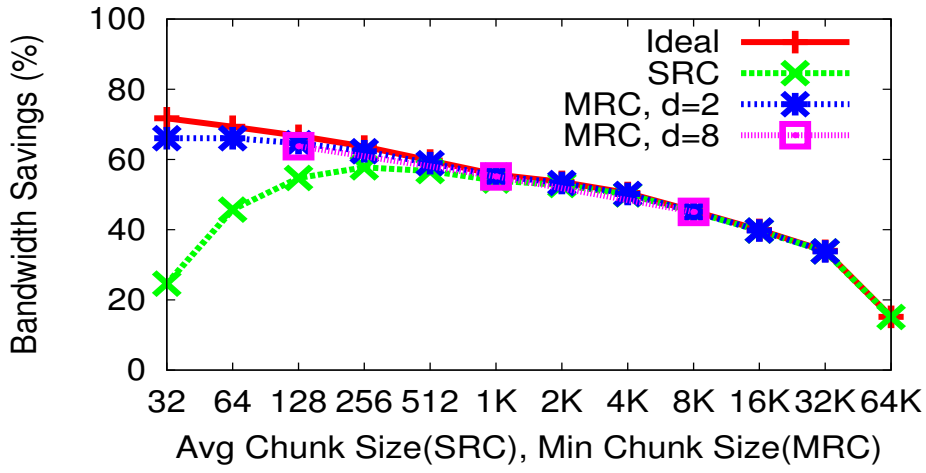
size of each tar file is about 276 MB, and the two files are 94% redundant. We refer to this workload as "Linux kernel".

Cacheability Breakdown Table 3.2 separates the potential bandwidth savings on the news sites by their HTTP cacheability, as determined by checking the cache control directives in the response headers. The top two numbers represent the portion of HTTP-uncacheable bytes (H-U), while the bottom two indicate HTTP-cacheable bytes (H-C). The middle two numbers show the portion of Wanax-cacheable bytes (W-C), while the outer two depict the Wanax-uncacheable portion (W-U).

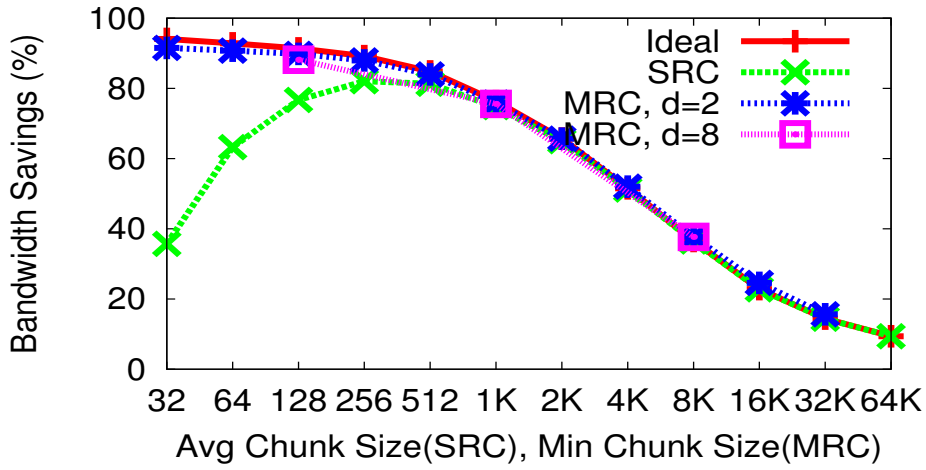
We see that most of the bytes are not cacheable by HTTP, but are cacheable by Wanax. Of the bytes that are not HTTP cacheable, about 75% are redundant and can benefit from Wanax. Of the HTTP-cacheable bytes, more than half are Wanax-cacheable as well. This result suggests that Wanax plus a browser cache can handle much of the traffic, but that Wanax with an HTTP proxy can provide even greater savings. Using an HTTP proxy with Wanax also allows HTTP-cacheable responses to be served directly from the proxy without re-contacting the content provider.

3.3.3 Results

Potential Bandwidth Savings Figure 3.7 shows the ideal and actual bandwidth savings on both workloads for various chunk sizes. As expected, the ideal bandwidth savings increases as the chunk size decreases. However, due to the chunk indexing



(a) News Sites

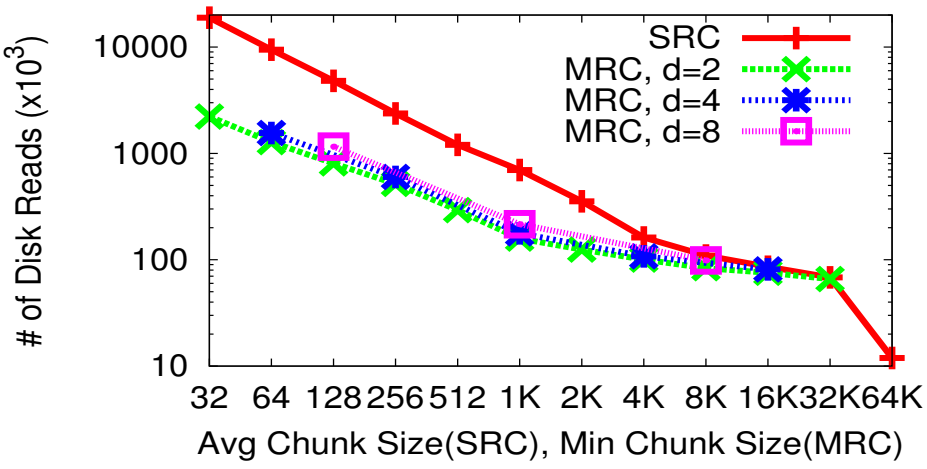


(b) Linux Kernel

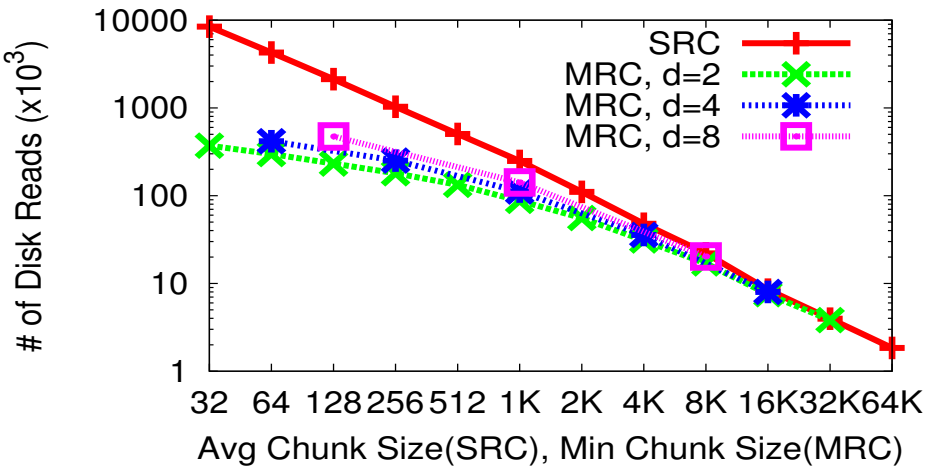
Figure 3.7: Potential Bandwidth Savings (d:degree) – SRC overheads prevent it from reaching ideal savings for smaller chunk sizes. MRC savings are close to ideal across all chunk sizes.

metadata transmission overhead, the actual savings with SRC peaks at a chunk size of 256 bytes with 58% bandwidth savings on the news sites, and 82% on the Linux kernel. The bandwidth savings drops as the chunk size further decreases, and when the chunk size is 32 bytes, the actual savings is only 25% on the news sites and 36% on the Linux kernel.

On the other hand, MRC approaches the ideal savings regardless of the minimum chunk size. With 32 byte minimum chunks, it achieves close to the maximum savings



(a) News Sites

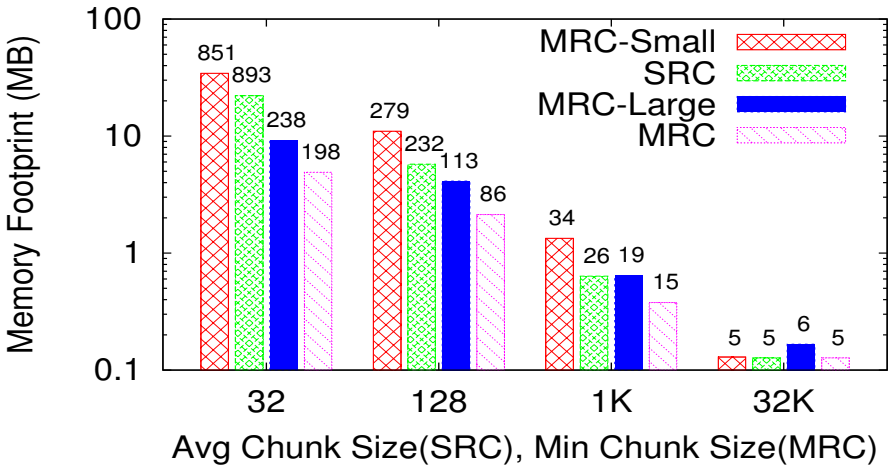


(b) Linux Kernel

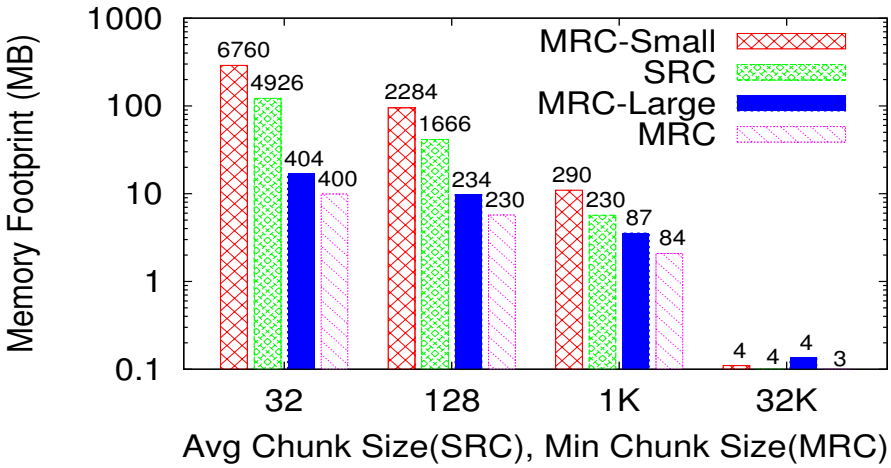
Figure 3.8: Disk Operation Cost (d:degree) – By using larger chunks when possible, MRC dramatically reduces the number of disk operations needed for a given workload. Note: Y axis is thousands of operations.

on both workloads – about 66% on the news sites and 92% on the Linux kernel. This is because MRC uses larger chunks whenever possible and the chunk name hint significantly reduces metadata transmission overheads. When comparing the best compression rates, MRC's effective bandwidth is 125% higher than SRC's on the Linux kernel while it shows 24% improvement on the news sites.

Disk Operation Cost MRC's reduced per-chunk indexing overhead becomes clearer if we look at the number of disk I/Os for each configuration, shown in



(a) News Sites

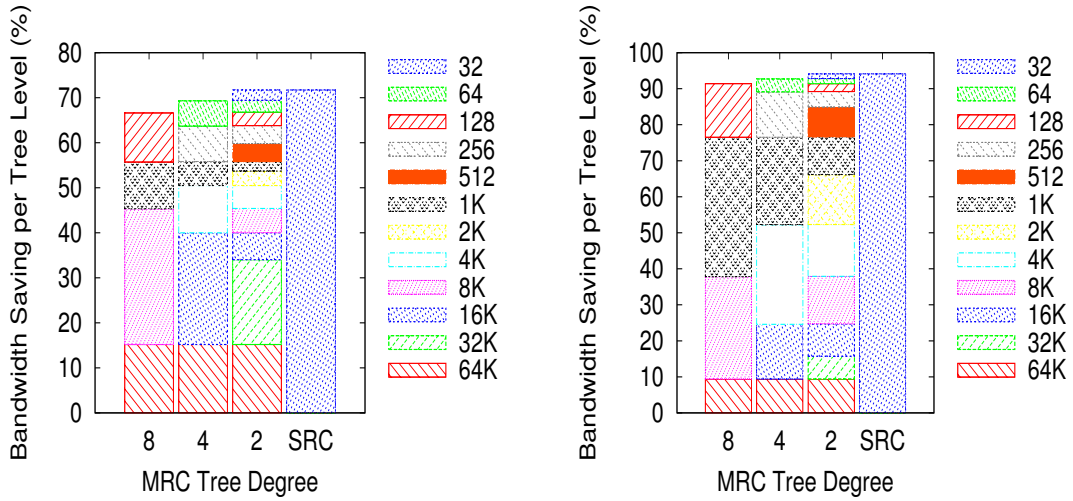


(b) Linux Kernel

Figure 3.9: Memory Footprint Comparison. Note log-scale Y axis. MRC's memory pressure is typically one-tenth that of SRC and MRC-Small. MRC-Large typically uses twice the memory due to backpointer overhead.

Figure 3.8. SRC's disk fetch cost increases dramatically as the chunk size decreases, making the use of small chunks almost impossible with SRC. MRC requires far fewer disk operations even at small chunk sizes. When the leaf node chunk size is 32 bytes, SRC performs 8.5 times as many disk operations on the news sites, and 22.7 times more on the Linux kernel.

Memory Pressure Memory pressure limits the amount of cache storage that a WAN accelerator can serve and the amount of RAM it requires for that storage.



(a) News Site (b) Linux Kernel

Figure 3.10: Per-level Bandwidth Savings in the MRC Tree – most MRC savings are from larger chunk sizes, reducing disk access and memory pressure.

Figure 3.9 compares the memory footprint with different chunking approaches. We count the number of chunk index entries that are used during the simulation, and calculate the actual memory footprint. Each bar represents the memory footprint (MB), and the numbers on top of each bar show the number of used cache entries in thousands. We show only the MRC trees with the degree 2, but other results follow the same trend.

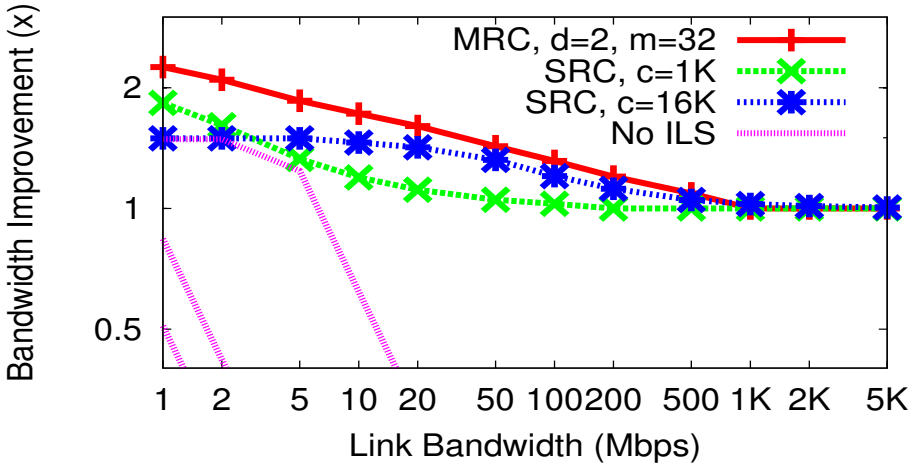
MRC incurs much less memory pressure than SRC does, since MRC requires one cache entry for any large chunk while SRC needs several cache entries for the same content. MRC-Small, however, requires even more cache entries than SRC does since reconstructing a larger chunk requires accessing all of its child entries. At a 32-byte chunk size, MRC-Small consumes almost 300 MB for the linux kernel while MRC requires only about 10 MB for the cache entries. MRC-Large shows a similar number of cache entries as MRC. However, the actual memory consumption of MRC-Large is much worse than MRC because every child chunk has a back pointer to its parent. MRC-Large consumes almost twice as much memory as MRC on the news workload.

MRC Chunk Size Breakdown Figure 3.10 shows the breakdown of bandwidth savings by different chunk sizes. We present all three MRC configurations and SRC with a 32-byte minimum chunk. For MRC, chunk sizes are sorted from smallest at top to largest at bottom, and the bottom bar shows the root chunk size of 64KB.

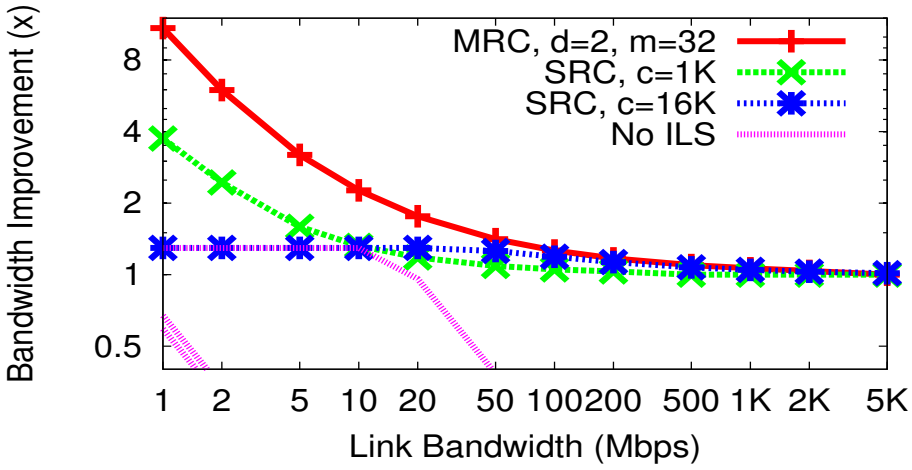
The results explain MRC's low disk overhead and low memory pressure – only a small fraction of the total savings comes from cache hits at the smallest chunks with MRC, whereas all of the savings is achieved by 32-byte chunk cache hits with SRC. Most of MRC's bandwidth reduction comes from larger chunks, which results in a much smaller number of disk I/Os and cache entries. We can see a similar trend across different MRC degrees. For example, the portion handled by a 4KB chunk size in MRC degree 4 is handled by 8KB chunk size as well in MRC degree 2. This means that some portion of 4KB chunks are merged into 8KB chunks in MRC degree 2. In all the MRC scenarios, chunks that are 4KB or larger provide 40-50% of the bandwidth savings, drastically reducing disk I/O.

Intelligent Load Shedding Based on the previous results of bandwidth savings and disk performance, we simulate the effective bandwidth improvement (times) given a target link capacity using ILS in Figure 3.11. We vary the link capacity from 1Mbps to 5Gbps, and assume one 7200RPM SATA disk.

We see that the effective bandwidth improvement of both MRC and SRC approaches one as link capacity increases, but SRC drops much faster than MRC. With smaller chunk sizes, SRC shows a high effective bandwidth with slow links due to its high compression rate, but the effective bandwidth quickly degrades as the link capacity grows. This is because with small chunks, the disk soon becomes the bottleneck of the system. In the same context, SRC with larger chunk sizes performs better with fast links, but shows a worse bandwidth improvement for slow links due to its low compression rate.



(a) News Sites



(b) Linux Kernel

Figure 3.11: Effective Bandwidth Improvement over Link Capacity (c: avg chunk size, d: degree, m: min chunk size) – as link capacity increases and disk performance becomes a bottleneck, MRC sheds cache hits on smaller chunks first, leading to a graceful degradation in effective bandwidth. With ILS disabled, the bandwidth collapses to the bottleneck disk speed. Note log-scale Y-axis.

MRC outperforms SRC regardless of link speed, and it sustains high effective bandwidth by leveraging multiple chunk sizes. If the link is slow, MRC fetches even the smallest chunks from disk, suppressing most redundancy. As the link capacity increases, MRC stops fetching the smaller chunks from disk, and focuses on the larger chunks rather than completely disabling compression, gracefully degrading the

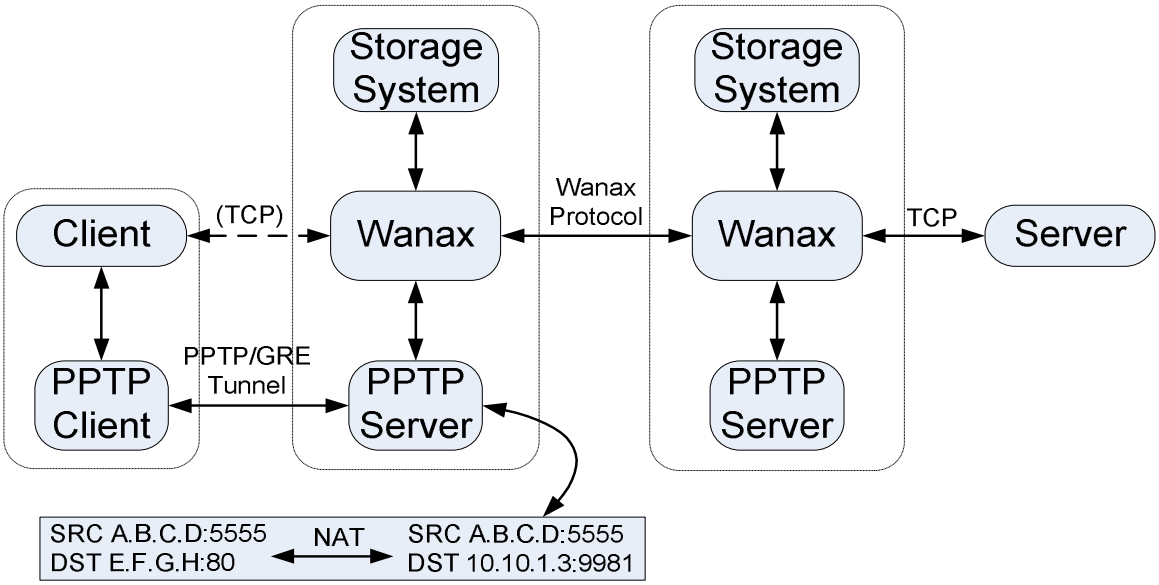


Figure 3.12: Wanax Implementation

effective bandwidth. When ILS is disabled, the effective bandwidth of all three configurations collapses to the bottleneck disk speed.

3.4 Implementation

The Wanax prototype consists of about 18,000 lines of C code sharing the same MRC/SRC code base with the simulator in Section 3.3. It uses an event-driven design with libevent [50], and has an architecture as shown in Figure 3.12.

PPTP/GRE Tunneling To provide easy access to end users, Wanax is implemented as an Internet gateway with PPTP/GRE tunneling, with TUN/TAP [102] support planned for the near future. Currently, users need to specify the IP address of Wanax in their PPTP client on Linux (or to set up a VPN client on Microsoft Windows), after which all traffic from the user is forwarded to the Wanax system. Wanax performs content fingerprinting only on TCP streams, and bypasses all non-TCP packets.

Reconstructing TCP Byte Streams While a fully transparent solution could intercept all IP packets and reconstruct TCP streams, that creates unnecessary complexity between layer 3 and 4. Instead, we intercept each TCP connection from the client, and redirect it to Wanax. This greatly simplifies the buffering process since Wanax can use the regular socket interface to recover the original content. We implement this in the PPTP server [77] by modifying the destination address and port of the incoming packets from the client, to those of Wanax. Similar to network address translation (NAT), we store this mapping in the address translation table, and recover the original address and port for the outgoing packets from Wanax to the client. This requires about 500 lines of PPTP server code modification.

Storage System We use HashCache [9] not only as an HTTP proxy, but also as scalable storage for storing and retrieving the chunk content as well as the chunk name hint. With a highly memory-efficient indexing scheme, HashCache fully utilizes a Terabytes-sized disk with less than 256 MB of physical memory, which is an ideal storage system for developing regions. HashCache is designed to use at most one disk seek for reading a random chunk, and performs group writes of related chunks to minimize disk latency for future reading. Wanax uses two special HashCache APIs, `hc_peek()` and `hc_hint()`. `hc_peek()` tells the existence of a chunk without performing actual disk I/O, and we use it for ILS and chunk name hints. `hc_hint()` exports the queuing status of the disk I/Os and is used for ILS calculations.

Optimizing Transport Protocol Inter-Wanax communication uses a set of techniques to improve network performance over high-latency WANs. While implementing a fully-custom transport protocol might yield some additional benefit, we opt for simplicity and use TCP variants optimized for high-delay, low-bandwidth links [33,37]. They modify the congestion avoidance algorithm so that they can quickly increase the congestion window even under high latency. In addition, Wanax multiplexes all com-

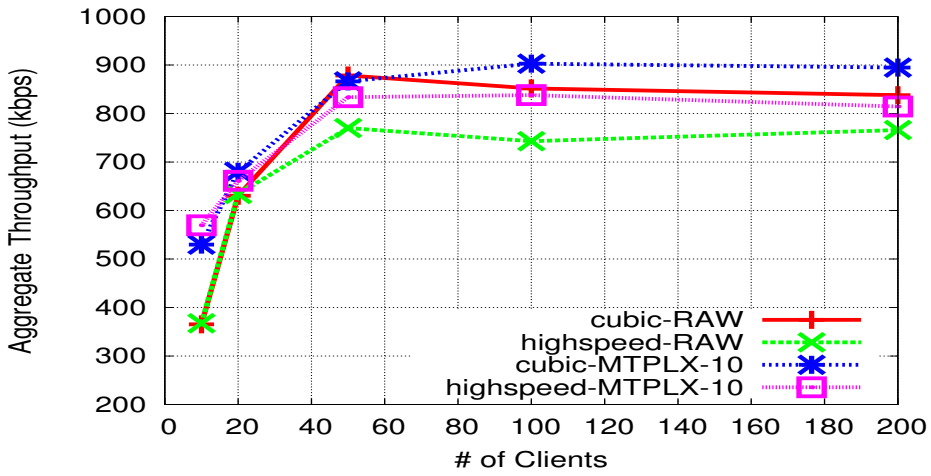


Figure 3.13: Multiplexing TCP Connections

munication over a set of long-lived TCP connections, avoiding an extra connection setup overhead of one RTT [68]. We also disable slow-start after idle time because we carefully control the number of connections per link. [11] These techniques are helpful especially for short-lived HTTP connections, which dominates traffic in the developing world [26]. In our tests, we find this combination yields close to the line speed even for many short connections.

Figure 3.13 shows the experiment result showing the effect of connection multiplexing. We capture the live CoDeeN [103] traffic, consisting of 1,000 objects, and compare the aggreagate throughput without multiplexing (RAW) against the one with multiplexing 10 connections (MTPLX-10), as a function of the concurrent number of clients. We run the replay test on a 1 Mbps link with 1,000 ms RTT on Emulab [104]. We compare CUBIC [37] and High Speed TCP [33] for congestion control algorithms, and plot the best throughput out of 3 trials. MTPLX-10 outperforms RAW when the number of clients is small due to RAW's connection setup overhead. MTPLX-10 shows the same trend even when the number of clients is large because the increased nummber of TCP connections in RAW leads to packet losses and the congestion degrades the aggregate throughput [61].

[11] sysctl '`tcp_slow_start_after_idle`' in Linux.

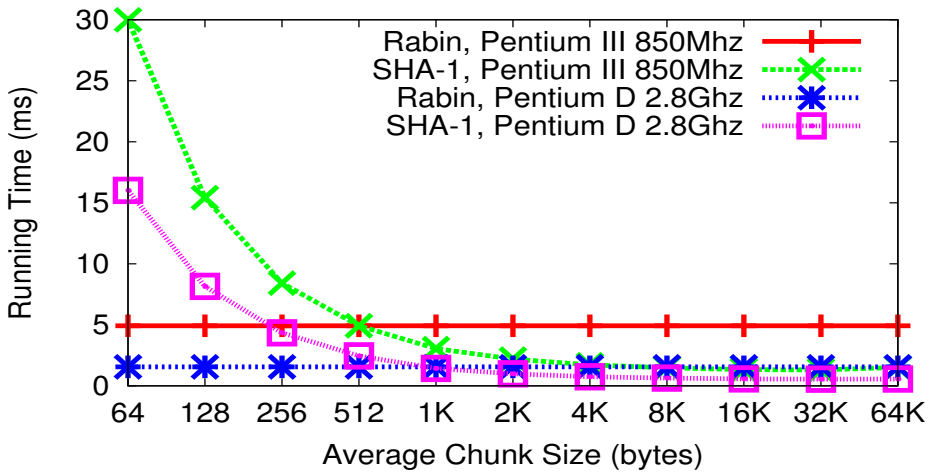


Figure 3.14: MRC Computation Overhead for 64KB Block

Minimizing MRC Computation Overhead While MRC preserves high bandwidth savings without sacrificing disk performance, it consumes more CPU cycles in fingerprinting and hash calculation due to an increased number of chunks. Figure 3.14 shows average time for running Rabin's fingerprinting algorithm and SHA-1 on one chunk with an average size of 64 KB from a 10 MB file. Surprisingly, Rabin's fingerprinting, though it is known to be computationally efficient, turns out to be still quite expensive, taking three times more than SHA-1. However, the aggregate SHA-1 cost increases as MRC's leaf chunk size decreases. If naively implemented, the total CPU cost of an MRC tree with a height n would be $n \times$ Rabin's fingerprinting time + sum of SHA-1 calculation of each level.

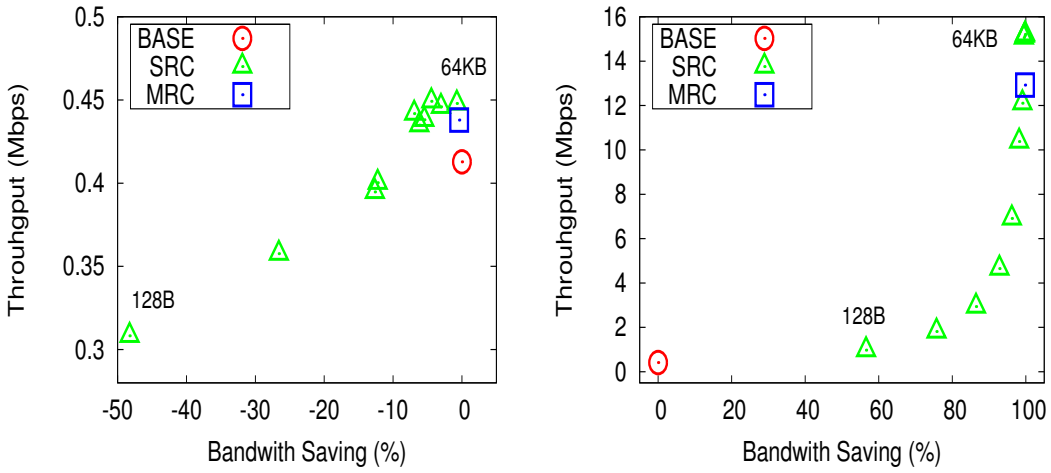
We consider two general optimizations which can be applied to both S-Wanax and R-Wanax. First, we run Rabin's fingerprinting on content only once, detect the smallest chunk boundaries, and derive the larger chunk boundaries from them. Second, we compute SHA-1 hashes only when necessary using the chunk name hint. For example, if S-Wanax knows that this chunk has been sent to R-Wanax before, S-Wanax assumes all of its children are already in R-Wanax and sends only the name of the parent. Likewise, if R-Wanax knows that a chunk has been stored on disk before, it does not re-store its children.

In addition, we implement an R-Wanax specific optimization. When the top-level chunk is a miss with R-Wanax but there are some chunk hits in the lower levels in the MRC tree, we only need to run fingerprinting with the cache-missed candidate list chunks. In order to support this, we now store a Rabin's fingerprint value (8 bytes) along with each chunk name hint. If a chunk in the candidate list is a cache hit, we can retrieve the fingerprint value for the chunk. If a chunk is a cache miss, we run the fingerprinting function to find and store any smaller chunks. We now know Rabin's fingerprint values for all chunks in the candidate list, so we can also reconstruct any parents without running the fingerprinting on the cache-hit chunks.

These optimizations are mainly for the case of chunk cache hits, where more CPU cycles are needed to deliver the chunks to the client. In case of a chunk cache miss, the bottleneck will still be in the slow WAN link for the developing world and consuming extra CPU cycles will not affect the download throughput.

3.5 Evaluation

In this section, we evaluate our prototype implementation of Wanax. Except for the realistic traffic test in the middle of this section, our tests use 1GHz AMD Athlon 64 X2 CPU machines equipped with 1GB RAM and a SATA disk. We divide them into two regions to represent the content provider and the developing region, with intra-region bandwidths set to 100Mbps. We vary the bandwidth and latency of the bottleneck WAN link connecting the two regions, depending on the evaluation scenarios. We have an origin server and an S-Wanax in the content provider side, and a client and two R-Wanax nodes in the developing region. Both the SRC and MRC tests are conducted using the same Wanax servers with the same TCP optimizations. To emulate the effect of large working sets which do not fit in memory, we disable in-memory cache for serving chunk content.

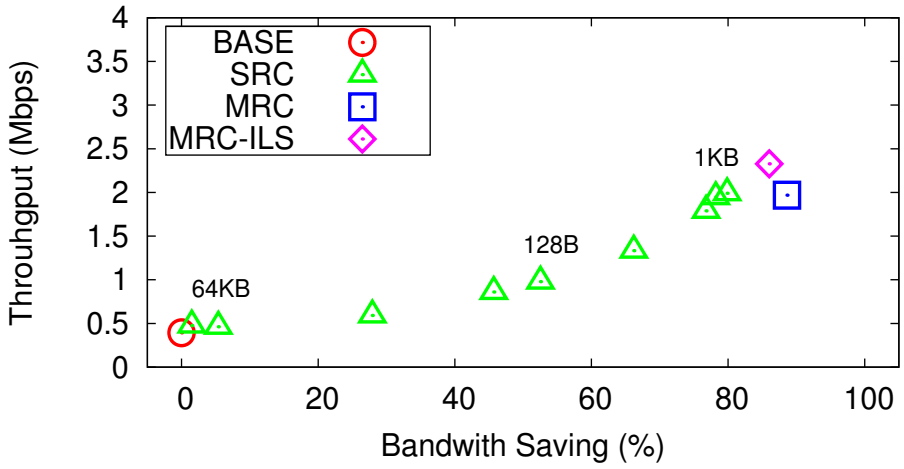


(a) 100% Cache Miss (b) 100% Cache Hit

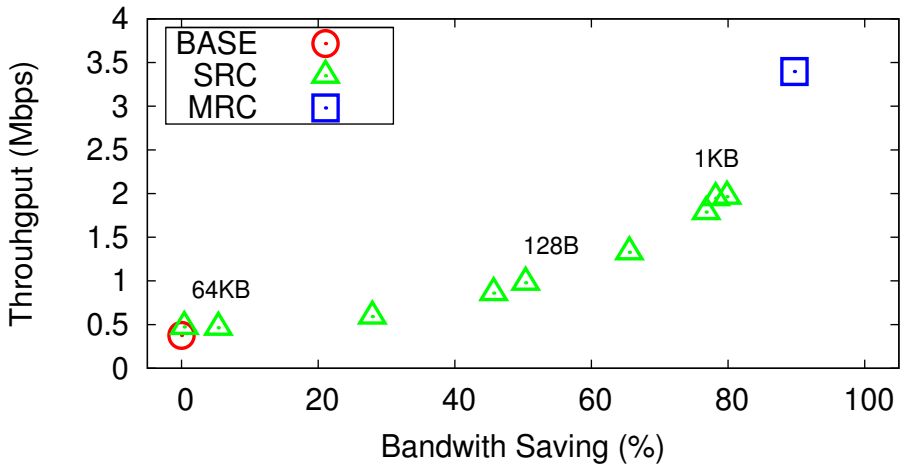
Figure 3.15: Cache Miss and Cache Hit Performance – even on all-hit or all-miss workloads, the extra overheads of MRC are small compared to SRC. The best SRC performers on this set use large chunk sizes, which would produce poor compression on realistic workloads.

Microbenchmark For our microbenchmark, we use two 1 MB files that have 90% redundancy using a 64-byte chunk size. The bottleneck WAN link is set to 512Kbps with a 200ms RTT. We download the first file twice to generate a cold cache miss and a complete cache hit, and then download the second file to generate a partial cache hit. We repeat the experiment by increasing the number of peers, and performing ILS. The downloading throughput (effective bandwidth) without Wanax (BASE) is only 0.41 Mbps due to the high WAN latency. We test SRC with chunk sizes from 128 bytes to 64KB, and a degree-8 MRC using a 128-byte minimum and 64KB maximum chunks.

Figure 3.15 (a) shows the bandwidth savings and throughputs when downloading the first file. Since every chunk is a cache miss, S-Wanax sends the content as well as the chunk name. Due to the chunk name overhead, SRC consumes more bandwidth than BASE, with up to 48% overhead for 128-byte chunks. However, the throughput is higher than BASE, reaching 0.45Mbps for 64KB chunks, due to the optimized TCP between Wanax nodes. On the other hand, the overhead of MRC is negligible since



(a) One node



(b) Two nodes

Figure 3.16: Performance with 90% redundancy and 512Kbps WAN link – MRC without ILS produces much better compression than any SRC configuration, and throughput is comparable to the best SRC. With ILS enabled, MRC produces better compression and throughput than any SRC configuration. When peering is used, disk is not a bottleneck, and enabling ILS has no effect.

it uses the largest chunk size of 64KB for most cache misses, yielding an overhead of 5.6% and a throughput of 0.43Mbps.

Figure 3.15 (b) compares MRC with SRC for a second download of the same file. As expected, SRC with the large chunk sizes (16, 32, and 64KB) shows the best throughput of 15Mbps. [12] As the chunk size decreases, the throughput degrades,

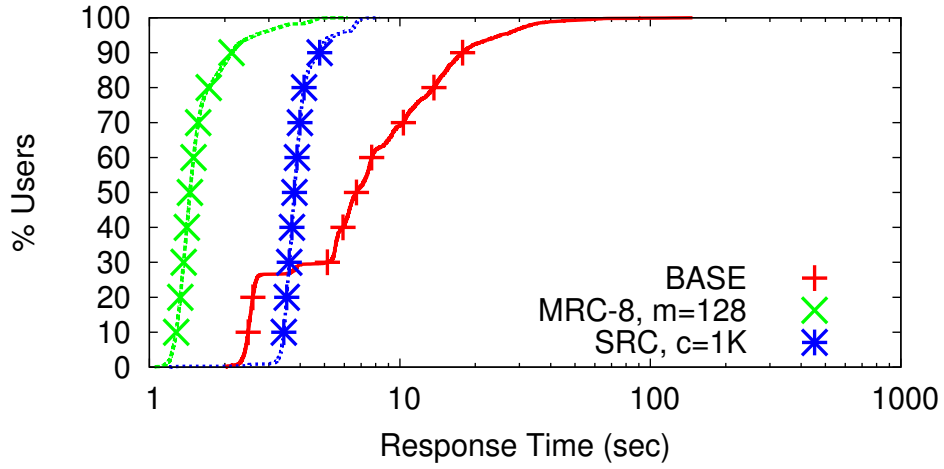
[12]The throughput is limited by the 200ms link latency since the total download time is 500-600ms. Downloading a larger file (10 MB) yields 44 Mbps throughput.

and the bandwidth savings is also reduced due to the per-chunk metadata overhead. However, MRC achieves both high throughput and bandwidth savings since they use the largest chunk size in this case. The slightly lower throughput of MRC versus SRC with large chunks is because MRC generates multiple chunk sizes for the first download, spreading the layout of the large chunks on disk, whereas the SRC download stores all of the chunks in sequence on disk.

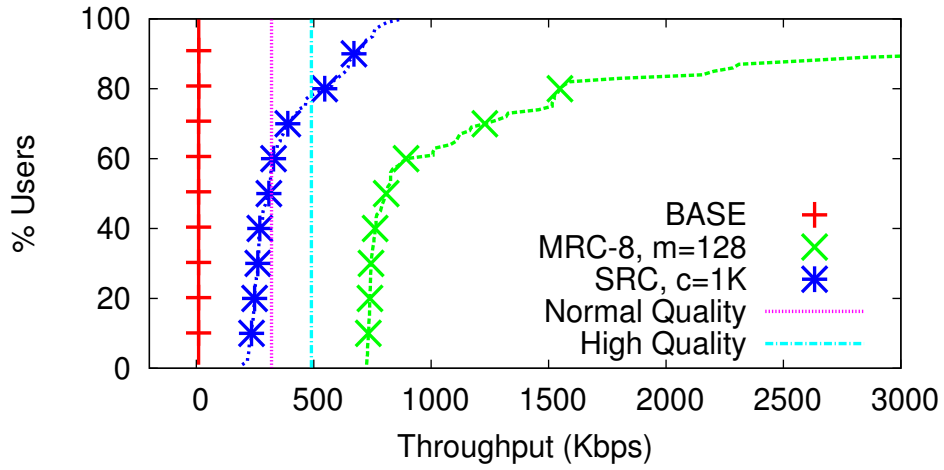
Figure 3.16 (a) depicts the performance of downloading the second file after warming the cache with the first file (90% redundancy). In this particular workload, SRC with 1KB chunks is the best configuration achieving both the highest bandwidth savings (80%) and highest throughput (2Mbps). MRC, in comparison, provides a higher bandwidth savings (89%) than any SRC scheme, but without ILS, the disk becomes the bottleneck and the throughput is almost the same as the best SRC. Enabling ILS raises the MRC throughput to 2.4Mbps at the cost of bandwidth savings, but beats every SRC configuration on both bandwidth savings and throughput – ILS automatically finds the sweet spot regardless of the workload.

Figure 3.16 (b) presents the effect of peering. The experiment is the same as the previous test, but now includes another Wanax peer in the developing region. Since peering allows Wanax to access multiple disks in parallel, we can expect improved throughputs by mitigating the disk bottleneck. However, for SRC, the lower compression rate causes the WAN bandwidth to be the bottleneck, so peering does not help. In comparison, MRC benefits significantly from peering, achieving 3.4Mbps throughput. With disk no longer the bottleneck, ILS is not necessary, and enabling it does not shed any load.

Realistic Traffic To test more general Web browsing in the developing regions, we use Alexa Top Sites [4] and YouTube [107] for testing using realistic traffic. We use the "pc850" nodes on Emulab [104], each equipped with an 850MHz Pentium III CPU



(a) Alexa



(b) YouTube

Figure 3.17: Realistic Traffic – both MRC and SRC provide compression on the Alexa workload, but MRC's median response time is 1.5 seconds, compared to 3.8 for SRC. For the YouTube test, all students would be able to view the video without interruption using MRC, while with SRC, it would be 20% for the high-quality version and 50% for the low-quality version.

and 512MB RAM. The bottleneck WAN link is set to 1Mbps with a 1000ms RTT, mimicking a satellite link commonly found in the developing world. First, we collect packet-level traces from Alexa's top 10 sites for Ghana and Nigeria, to reflect common Web browsing activity in these regions, including both cacheable and uncacheable objects. We replay 5,000 connections with 200 simultaneous clients on the traffic, and measure the response time. We also pick one of the most popular videos at the

time of testing [13] from YouTube, and have 100 clients simultaneously download the whole 18 MB clip. YouTube's video content is not cacheable by standard Web proxies since its URL is in a customized format and changes for each download. This test is intended to reflect a classroom scenario where a number of students watch the same clip roughly at the same time. We introduce an 1 second interval between the client requests, and measure the throughput of each transfer. For these experiments, we use only one R-Wanax, configured with either a degree-8 MRC tree or a 1 KB SRC configuration, which has shown good performance and bandwidth savings.

Figure 3.17 (a) shows the response time CDFs for the Alexa workload. The average object size is 5,425 bytes and the median is 570 bytes. MRC outperforms both SRC and direct transfer (BASE), and shows the median response time of 1.5 seconds while BASE and SRC show 6.7 and 3.8 seconds each. MRC and SRC are generally faster than BASE because they fetch most objects from the local disk cache. However, on this workload, MRC typically uses one disk read per object while SRC frequently uses multiple disk I/Os per object. This behavior explains the performance difference between the two, and the disk latency sometimes makes SRC worse than BASE.

Figure 3.17 (b) shows the YouTube results. The bitrate of the video is 490 Kbps and the BASE curve shows that nobody would be able to watch the clip reliably on a 1 Mbps link. SRC would satisfy only about 20% of the users while MRC would deliver the video to all 100 clients without interruption. The median throughputs are 809 Kbps and 309 Kbps for MRC and SRC each. We test the lower quality video (320 Kbps) of the same content, and find that SRC satisfies half of the users. Without a WAN accelerator, only two or three clients can watch the clip at any given time, which makes using classroom video problematic.

Enterprise Environment Finally, we evaluate Wanax in an enterprise-like environment to determine how well it performs compared to commercial WAN accel-

[13] The first weekly address by President Obama on 01/24/09

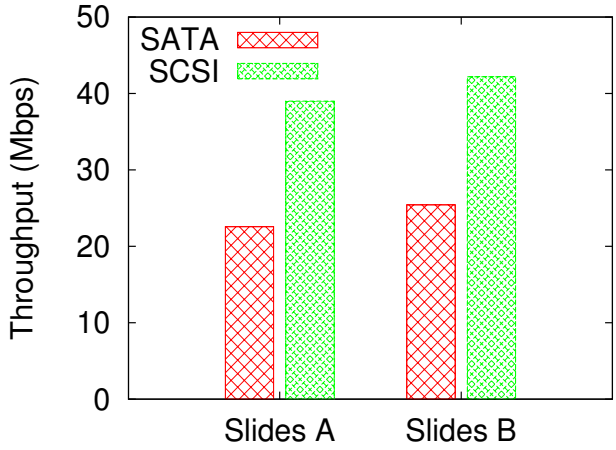


Figure 3.18: Enterprise Environment – with no link bottleneck, the underlying system performance can be measured. No standard test exists for these systems, but these figures are comparable to those published for commercial systems.

erators. Unfortunately, while vendors publish performance figures, none appear to publish the test scenarios they use. Testing in industry magazines uses LANs to remove network capacity as the bottleneck, which we also use in this test. That is, we focus on the impact of disk performance by separating the network delay from the overall throughput. This is because the disk performance is the bottleneck in higher link capacity enterprise environments. A high-end commercial product targeting large offices or data centers uses multiple small capacity SCSI disks, [14] rather than one large capacity disk [85].

We create three sets of PowerPoint slides – an original deck that is 11.9 MB, and two modified decks that add slides to this deck, yielding a 13.9 MB file (Slides A) and a 14.9 MB file (Slides B). Compared with the original deck, these have redundancies of 86% and 81%. This represents a scenario where multiple people in different offices are collaborating on a presentation. We first warm the Wanax cache with the original file, and measure the throughput of the two modified slide decks. The measured bandwidth savings correspond to the redundancy in the files. We use MRC degree

[14] 1U product supporting 45Mbps uses 4 disks, 3U product supporting 310Mbps uses 16 disks.

8 with the minimum chunk size of 128 bytes, and repeat the experiments with two different disks.

As shown in Figure 3.18, both file downloads achieve slightly more than 20 Mbps with a single 7200 RPM SATA disk at R-Wanax. The slightly larger redundancy of Slides A (86%) incurs more disk hits than Slides B (81%), and it is reflected in B's slightly larger throughput. With a faster 15K RPM SCSI disk, the throughput almost doubles in both cases. In examining the configurations of one of the leading WAN accelerator companies [85], we see that their per-disk performance ranges from 8 Mbps to 20 Mbps depending on the configuration. Since we have incomplete information about the testing scenario, we cannot draw any firm conclusions, but our range of 20-40 Mbps suggests that we have at least comparable performance to commercial solutions in these higher-end configurations, and our memory pressure analysis suggests that Wanax does so using a small fraction of the memory of these systems.

3.6 Related Work

Much work, both commercial and academic, has been done in the broad area of redundancy elimination for network traffic. Web caching has been an active field, with the first-generation caches [17, 54] storing unchanging objects in their entirety, often with protocol support. Later techniques included delta encoding [59] to reduce traffic for object updates, and duplicate detection to suppress downloading of aliased HTTP objects [60].

Spring and Wetherall [94] further extend the previous approaches to sub-packet granularity, and develop a protocol-independent content fingerprinting (CF) scheme that eliminates redundancy over a single link. Recently, Anand *et al.* [5] extend this idea on ISP routers, with an emphasis on redundancy-aware routing algorithms.

RTS-id [2] also eliminates redundancy in the wireless environment by caching recently transferred packets through eavesdropping. However, they all work on a per-packet basis at the link layer, which limits the potential bandwidth saving to the packet size. Since Wanax operates on byte streams, it does not have such limits.

Content fingerprinting has been widely adapted in many applications, including network file systems [7, 64], Web proxies [16, 82], file transfer services [78, 79], and Web servers [70]. However, all of these systems are application-specific, and do not work across protocols. DOT [101] proposes a flexible architecture for generic data transfer, which is protocol independent, but it requires application-level modification. Ditto [25] extends DOT, and targets wireless mesh network environments. It is complementary to Wanax since Wanax focuses on eliminating redundancy on the bottleneck WAN link.

There are a number of commercial WAN accelerators [21, 84, 92] as well. They operate below the application layer, so they are both transparent and protocol independent. However, they are designed to run on dedicated server-class appliances with fast disks and a large pool of memory. Also, their typical enterprise deployment scenario is a star topology where branch offices are speaking only to a central office. Running them on the resource-limited shared machines with a mesh topology in the developing world would be problematic, and lead to poor performance even if possible. Instead, Wanax is designed from the scratch to specifically address the developing world's needs, and we believe some of our techniques such as MRC and ILS can also be applied to the enterprise scenarios to reduce the deployment cost.

To the best of our knowledge, Wanax is the first system to simultaneously use multiple chunk sizes. Riverbed [84] uses a bottom-up segmentation scheme [58] that first uses 100 byte chunks, and then creates larger pseudo-chunks that contain the names of the smaller chunks [83], which is similar to MRC-Small. This approach provides some of the disk efficiency and bandwidth benefits of MRC, but still requires

access to all of the metadata of the 100-byte chunks, thereby retaining the memory pressure of the smaller chunks. In the context of large file replication, Remote Differential Compression [99] uses a similar recursive segmentation scheme with a minimum chunk size of 1 KB, in order to reduce the size of chunk names sent over the network. Most recently, multi-resolution handprinting [97] proposes an efficient technique for choosing the best chunk sizes for the given similar files, by comparing handprints - a deterministic subset of chunk hashes with different chunk sizes. We share the same spirit of exploiting trade-offs of multiple chunk sizes. However, their method is based on static analysis on the files they already have. MRC is a dynamic counterpart, and is directly applicable for online processing.

Finally, there are a number of active research projects for the developing world. DitTorrent [88] shares the same idea of exploiting better regional connectivity as Wanax, but focuses on scheduling P2P dialup connections. As systems like rural WiFi [72] or WiMAX [105] extend the Internet to new regions, Wanax can help improve the effective bandwidth delivered.

3.7 Summary

We have presented the design and implementation of Wanax, a flexible and scalable WAN accelerator targeting developing regions. Using a novel chunking technique, MRC, Wanax provides high compression and high throughput, while maintaining a small memory footprint. This profile enables it to run on resource-limited shared hardware, an important requirement in developing-world deployments. By exploiting MRC to direct load shedding, Wanax is designed to maximize the effective bandwidth even when disk performance is poor due to overloading. The peering scheme used in Wanax allows multiple servers in a region to share their resources, and thereby exploit faster and cheaper local-area connectivity instead of always using the WAN. In

summary, through a careful design addressing the developing world challenges, Wanax provides customized, cost-effective WAN acceleration to the region with commodity hardware. We have begun deploying Wanax at a few partner sites in Africa, and expect to have more results about real-world operation in the future.

Chapter 4

Conclusions and Future Work

The Web is clearly one of the most important Internet applications, and many people's daily lives rely heavily on it. Despite its importance, we still face limited Web access problems because of the following two trends. First, the Web has changed and grown significantly, requiring more and more bandwidth, especially due to the increasing popularity of video streaming sites. Second, as the need for Web access has also grown, a large fraction of users in bandwidth-limited environments, such as people in the developing world or mobile device users, still suffer from poor Web access. While there has been a burst of research in the past decade aimed at understanding the nature of underlying Web traffic and thus improving Web access, unfortunately, it has dropped off just as the Web has changed significantly. Essentially, there is a gap between previous studies and today's Web, which motivates this dissertation.

Towards the goal of improving Web access, this thesis has attempted to answer the following questions.

- What has changed in Web traffic over time?
- How can we analyze today's dynamic Web pages that involve complex client-side interactions such as Ajax?

- What is the implication of content-based caching on today's real Web traffic, such as the effective byte hit rate and required cache storage size?
- How can we design a content-based caching system that is suitable for resource-limited developing world environments?

We have addressed the first three questions in Chapter 2, and the last question in Chapter 3.

4.1 Understanding Modern Web Traffic

For a better understanding of today's Web traffic, in Chapter 2, we have analyzed 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. Our large-scale data set is unique in that it spans many years, covers a world-wide user population, and includes the full request-response content instead of just access logs.

Among our major findings, we observe a rise of Flash video and Ajax traffic, which is also correlated with the increasing object size of JavaScript and CSS, and the increased number of concurrent connections from Web browsers. Also, we found that search engine and analytics sites reach an increasingly large fraction of users (up to 65%), which has implications for user tracking and privacy.

In addition, our new Web page analysis technique called StreamStructure 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, we find that the pages have grown larger in terms of both the number of objects and size, but the page loading latency has decreased due to the increased number of concurrent browser connections and improved caching behavior.

Finally, we observe that an intelligent use of large cache storage could provide almost twice the byte hit rate of traditional object-based caching, and is also effective

for aborted transfers which are mostly video. We also quantify the origins of redundancy, and find that most of the additional savings of content-based caching are due to the partial content overlaps.

4.2 Improving Modern Web Traffic Caching

Motivated by the potential benefits of content-based caching approaches from our analysis, in Chapter 3, we present the design and implementation of Wanax, a flexible and scalable WAN accelerator specifically targeting resource-constrained developing regions.

The design of Wanax has the following three contributions. First, we have developed a novel chunking scheme called multi-resolution chunking (MRC). By using multiple sized chunks simultaneously, MRC provides high compression rate and high disk throughput at the same time, while maintaining a small memory footprint. Second, we have developed an intelligent load shedding (ILS) scheme that exploits MRC. By carefully balancing network and disk usage, ILS maximizes the effective bandwidth even when disk performance is poor due to overloading. Finally, Wanax is designed to exploit mesh network environments in developing regions. By sharing resources among multiple servers in a region, Wanax enables aggregation of disk space, parallel disk access, and efficient use of faster and cheaper local bandwidth, instead of always using the WAN.

We have also implemented a prototype of Wanax, and demonstrated that Wanax not only enables smooth video streaming but also reduces response time of Web browsing in low-bandwidth environments. In addition, when equipped with server-class hardware, Wanax provides comparable performance to enterprise WAN accelerators, but with a small memory footprint.

4.3 Future Work

There are many open issues along the line of research in this dissertation. We discuss several future directions that deserve further investigations.

New prefetching technique Prefetching is a well-known technique to reduce user-perceived latency by downloading objects in advance that are likely requested by users. Unlike caching that works passively on-demand, prefetching proactively predicts future requests at the cost of extra network traffic from wrong guesses. Due to the potential benefit of prefetching, browsers have begun to employ prefetching in one way or another [19, 32]. While many schemes have been proposed in the past decade [27, 30, 48, 56, 69], they essentially rely on users' access history for prediction – they can prefetch known objects only. This becomes problematic for today's Web where more and more previously unseen objects are continuously generated. Indeed, we observed that the fraction of such objects is growing, and they account for up to 90% of the entire objects in 2010 as shown in Section 2.4. Therefore, it would be interesting to develop a new prefetcing scheme that could predict previously unseen objects. While it looks challenging, it would greatly reduce Web page loading latency as discussed in Section 2.3. Another intriguing direction is to investigate the case for combining content-based caching with prefetching. This will allow more aggressive prefetching as content-based caching can reduce network bandwidth consumption.

Analysis of new video streaming technologies Video traffic has seen a rapid growth, and one study expects that it will account for 61% of the Internet traffic by 2015 [20]. To accommodate this huge volume of traffic, several new video streaming technologies called *adaptive streaming over HTTP* [1, 8, 63] were proposed in 2009 and 2010. Similar to content-based caching, adaptive streaming splits a large video file into multiple smaller chunks for cacheability and performance. In addition, it

switches video quality in real-time by dynamically monitoring local bandwidth and video rendering performance. Despite the potential benefits from adaptive streaming, these relatively new technologies are not fully understood yet and thus require careful assessment. For example, it would be interesting to investigate the effectiveness of adaptive streaming, such as its actual cache hit rate, bandwidth savings, and required cache storage size. Another possible direction is to examine the interaction between adaptive streaming and content-based caching. Whether they impact each other positively or negatively remains an open question.

Precise analysis of Web page loading latency Page loading latency is a very important metric that directly affects users' browsing experience as well as the traffic and revenue of Web sites. Recently, search engines such as Google incorporate the page loading speed into their page ranking algorithm, giving a higher rank to the faster Web sites as an incentive for optimizing page loading latency [35]. Despite its importance, page loading latency is not fully understood because it is determined by many factors including DNS lookup, object downloading time, server processing time, and browser parsing/rendering time. Furthermore, today's Web pages consist of many third-party components such as advertisement network and analytics sites, but it is unclear how much they contribute to the page loading latency. While this dissertation briefly investigates this issue with access log data sets in Section 2.3, it would be more interesting if we could analyze the page loading process precisely with more detailed information. For example, we could instrument browsers or proxies to capture all meaningful events during the page loading process. Another possible direction is to analyze publicly available data sets that have much more detailed information than simple access logs [39].

Network acceleration on mobile devices Mobile devices has become an essential part of our lives and a popular means to access the Internet. Unfortunately,

the wireless network bandwidth (3G/4G/WiFi) of mobile devices is still very poor, and it severely limits the potential benefit and application of the always-on Internet connectivity of mobile devices. While deploying WAN accelerators between mobile devices and remote servers can reduce bandwidth consumption in the WAN link, the last-mile link near mobile devices still remains a bottleneck. Thus, it would be interesting to investigate the case for incorporating network acceleration on mobile devices themselves. Unlike network acceleration on proxy machines, this case opens up multiple new challenges: (1) the cache storage of mobile devices is very limited – how to manage the limited storage while achieving high cache hit rates is an interesting question to investigate, (2) users are mobile – assuming that WAN acceleration boxes are already widely deployed, how to synchronize the cache with different boxes as users move will be an essential part of the research, and (3) a battery is another scarce resource on mobile devices – network acceleration should be performed carefully not so as to drain the battery.

Unified framework for network acceleration Network acceleration is a popular and effective technique to relieve a network bottleneck and is used in numerous systems such as file systems, storage systems, Web servers, and network accelerators. At the core of network acceleration is data deduplication – to identify redundant data and avoid sending them, which requires schemes for data segmentation (chunk generation), indexing, and storing content. Unfortunately, while virtually all network acceleration systems exploit data deduplication under the hood, they are built separately from the scratch with different storage options and schemes, due to different workloads and application requirements. If one wants to utilize a new storage medium such as solid-state disk [75] (SSD) or phase-change memory [81] (PCM), one needs to redesign and rebuild everything. So, it would be useful to develop a unified framework for network acceleration, which relieves such burdens. The end result will

be a flexible and adaptive middleware or proxy that performs network acceleration given a specification of application requirements and resource budget. For example, one could specify the amount of memory, flash/SSD and disk, network bandwidth, and the number and speed of CPU cores.

Bibliography

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

[2] Mikhail Afanasyev, David G. Andersen, and Alex C. Snoeren. Efficiency through eavesdropping: Link-layer packet caching. In *Proc. 5th USENIX NSDI*, San Francisco, CA, April 2008.

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

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

[5] Ashok Anand, Archit Gupta, Aditya Akella, Srinivasan Seshan, and Scott Shenker. Packet caches on routers: The implications of universal redundant traffic elimination. In *Proc. ACM SIGCOMM*, Seattle, WA, August 2008.

[6] Ashok Anand, Chitra Muthukrishnan, Aditya Akella, and Ramachandran Ramjee. Redundancy in network traffic: Findings and implications. In *Proc. ACM SIGMETRICS*, Seattle, WA, June 2009.

[7] Siddhartha Annapureddy, Michael J. Freedman, and David 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] Anirudh Badam, KyoungSoo Park, Vivek S. Pai, and Larry L. Peterson. Hashcache: Cache storage for the next billion. In *Proc. 6th USENIX NSDI*, Boston, MA, April 2009.

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

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

[12] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. Web caching and zipf-like distributions: Evidence and implications. In *Proc. IEEE INFOCOM*, New York, NY, March 1999.

[13] Ramón Cáceres, Fred Douglis, Anja Feldmann, Gideon Glass, and Michael Rabinovich. Web proxy caching: the devil is in the details. In *Proc. 1st ACM Workshop on Internet Server Performance*, Madison, WI, June 1998.

[14] Tom Callahan, Mark Allman, and Vern Paxson. A longitudinal view of HTTP traffic. In *Passive & Active Measurement (PAM)*, Zurich, Switzerland, April 2010.

[15] Meeyoung Cha, Haewoon Kwak, Pablo Rodriguez, Yong-Yeol Ahn, and Sue 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, October 2007.

[16] R. Chakravorty, A. Clark, and I. Pratt. Optimizing web delivery over wireless links: Design, implementation and experiences. *IEEE JSAC*, 2003.

[17] A. Chankhunthod, P. Danzig, C. Neerdaels, M.F. Schwartz, and K.J. Worrell. A Hierarchical Internet Object Cache. In *Proc. USENIX Annual Technical Conference*, San Diego, CA, January 1996.

[18] Hyoung-Kee Choi and John O. Limb. A behavioral model of web traffic. In *IEEE International Conference on Network Protocols (ICNP)*, Toronto, Canada, October 1999.

[19] Chromium Blog: Prerendering in Chrome. `http://blog.chromium.org/2011/06/prerendering-in-chrome.html`.

[20] Cisco Visual Networking Index (VNI) Forecast (2010-2015). `http://www.cisco.com/en/US/netsol/ns827/networking_solutions_sub_solution.html#~forecast`.

[21] Citrix Systems. `http://www.citrix.com/`.

[22] Dave Crane, Eric Pascarello, and Darren James. *Ajax in Action*. Manning Publications Co., Greenwich, CT, USA, 2005.

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

[24] Data Domain. `http://www.datadomain.com/`.

[25] Fahad Dogar, Amar Phanishayee, Himabindu Pucha, Olatunji Ruwase, and David Andersen. Ditto - a system for opportunistic caching in multi-hop wireless mesh networks. In *Proc. ACM MobiCom*, San Francisco, CA, September 2008.

[26] Bowei Du, Michael Demmer, and Eric Brewer. Analysis of WWW traffic in Cambodia and Ghana. In *Proc. Fifteenth International World Wide Web Conference*, Edinburgh, Scotland, May 2006.

[27] Dan Duchamp. Prefetching hyperlinks. In *Proc. 2nd USENIX Symposium on Internet Technologies and Systems (USITS)*, Boulder, CO, October 1999.

[28] Jeffrey Erman, Alexandre Gerber, Mohammad T. Hajiaghayi, Dan Pei, and Oliver Spatscheck. Network-aware forward caching. In *Proc. 18th International World Wide Web Conference*, Madrid, Spain, May 2009.

[29] Li Fan, Pei Cao, Jussara Almeida, and Andrei Z. Broder. Summary cache: A scalable wide-area Web cache sharing protocol. In *Proc. ACM SIGCOMM*, pages 254–265, Vancouver, British Columbia, Canada, September 1998.

[30] Li Fan, Pei Cao, Wei Lin, and Quinn Jacobson. Web prefetching between low-bandwidth clients and proxies: potential and performance. In *Proc. ACM SIGMETRICS*, Atlanta, GA, June 1999.

[31] 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.

[32] Link prefetching FAQ – MDN Docs. `https://developer.mozilla.org/en/Link_prefetching_FAQ`.

[33] S. Floyd. *HighSpeed TCP for Large Congestion Windows.* United States, 2003. RFC 3649.

[34] Frank La Rue. Report of the Special Rapporteur on the promotion and protection of the right to freedom of opinion and expression. *United Nations General Assembly Human Rights Council*, May 2011.

[35] Official Google Webmaster Central Blog: Using site speed in web search ranking. `http://googlewebmastercentral.blogspot.com/2010/04/using-site-speed-in-web-search-ranking.html`.

[36] 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, December 1997.

[37] Sangtae Ha, Injong Rhee, and Lisong Xu. Cubic: a new TCP-friendly high-speed TCP variant. *Operating Systems Review*, 42(5), 2008.

[38] F. Hernandez-Campos, K. Jeffay, and F.D. Smith. Tracking the evolution of web traffic: 1995-2003. In *Proc. IEEE/ACM MASCOTS*, oct 2003.

[39] HTTP Archive. `http://httparchive.org/`.

[40] Sunghwan Ihm and Vivek S. Pai. Towards understanding modern web traffic. In *Proc. Internet Measurement Conference (to appear)*, Berlin, Germany, November 2011.

[41] Sunghwan Ihm and Vivek S. Pai. Towards understanding modern web traffic (extended abstract). In *Proc. ACM SIGMETRICS*, San Jose, CA, June 2011.

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

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

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

[45] D. L. Johnson, Elizabeth M. Belding, Kevin Almeroth, and Gertjan 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.

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

[47] Terence Kelly and Jeffrey Mogul. Aliasing on the world wide web: prevalence and performance implications. In *Proc. Eleventh International World Wide Web Conference*, Honolulu, Hawaii, USA, May 2002.

[48] Thomas M. Kroeger, Darrell D. E. Long, and Jeffrey C. Mogul. Exploring the bounds of web latency reduction from caching and prefetching. In *Proc. 1st USENIX Symposium on Internet Technologies and Systems (USITS)*, Monterey, CA, December 1997.

[49] Craig Labovitz, Scott Iekel-Johnson, Danny McPherson, Jon Oberheide, and Farnam Jahanian. Internet inter-domain traffic. In *Proc. ACM SIGCOMM*, New Delhi, India, August 2010.

[50] Libevent. `http://monkey.org/~provos/libevent/`.

[51] Libnids. `http://libnids.sourceforge.net/`.

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

[53] Gregor Maier, Anja Feldmann, Vern Paxson, and Mark Allman. On dominant characteristics of residential broadband internet traffic. In *Proc. Internet Measurement Conference*, Chicago, Illinois, November 2009.

[54] Carlos Maltzahn, Kathy J. Richardson, and Dirk Grunwald. Performance issues of enterprise level web proxies. In *Proc. ACM SIGMETRICS*, Seattle, WA, June 1997.

[55] Udi Manber. Finding similar files in a large file system. In *Proc. Winter USENIX Conference*, pages 1–10, San Francisco, CA, January 1994.

[56] Evangelos P. Markatos and Catherine E. Chronaki. A top-10 approach to prefetching on the web. In *In Proceedings of the Annual Conference of the Internet Society*, 1998.

[57] MaxMind. `http://www.maxmind.com/`.

[58] Steven McCanne and Michael J. Demmer. US patent #7,116,249: Content-based segmentation scheme for data compression in storage and transmission including hierarchical segment representation, 2006.

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

[60] Jeffrey C. Mogul, Yee Man Chan, and Terence Kelly. Design, implementation, and evaluation of duplicate transfer detection in HTTP. In *Proc. 1st USENIX NSDI*, San Francisco, CA, March 2004.

[61] Robert Morris. TCP Behavior with Many Flows. In *IEEE International Conference on Network Protocols (ICNP)*, October 1997.

[62] Firefox web browser. `http://www.mozilla.com/firefox/`.

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

[64] Athicha Muthitacharoen, Benjie Chen, and David Mazieres. A low-bandwidth network file system. In *Proc. 18th ACM Symposium on Operating Systems Principles (SOSP)*, Banff, Canada, October 2001.

[65] NIST. Secure Hash Standard (SHS). In *FIPS Publication 180-1*, 1995.

[66] NTIA National Broadband Map reveals Internet deserts, oases in US. `http://www.digitaltrends.com/computing/ntia-national-broadband-map-reveals-internet-deserts-oases-in-us/`.

[67] One Laptop Per Child. `http://www.laptop.org/`.

[68] V. N. Padmanabhan and J. C. Mogul. Improving HTTP Latency. In *Proc. Second International WWW Conference*, October 1994.

[69] Venkata N. Padmanabhan and Jeffrey C. Mogul. Using predictive prefetching to improve world wide web latency. *ACM Computer Communications Review*, 26, January 2005.

[70] KyoungSoo Park, Sunghwan Ihm, Mic Bowman, and Vivek S. Pai. Supporting practical content-addressable caching with CZIP compression. In *Proc. USENIX Annual Technical Conference*, Santa Clara, CA, June 2007.

[71] KyoungSoo Park, Vivek S. Pai, Kang-Won Lee, and Seraphin Calo. Securing web service by automatic robot detection. In *Proc. USENIX Annual Technical Conference*, Boston, MA, June 2006.

[72] Rabin Patra, Sergiu Nedevschi, Sonesh Surana, Anmol Sheth, Lakshminarayanan Subramanian, and Eric Brewer. Wildnet: Design and implementation of high performance wifi based long distance networks. In *Proc. 4th USENIX NSDI*, Cambridge, MA, April 2007.

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

[74] Stefan Podlipnig and Laszlo Böszörmenyi. A survey of web cache replacement strategies. *ACM Computing Surveys*, 35, December 2003.

[75] M. Polte, J. Simsa, and G. Gibson. Enabling enterprise solid state disks performance. In *Proc. of Workshop on Integrating Solid-state Memory into the Storage Hierarchy*, Mar 2009.

[76] Lucian Popa, Ali Ghodsi, and Ion Stoica. HTTP as the Narrow Waist of the Future Internet. In *Proc. 9th ACM Workshop on Hot Topics in Networks (Hotnets-IX)*, Monterey, CA, October 2010.

[77] Poptop - The PPTP Server for Linux. `http://www.poptop.org/`.

[78] Himabindu Pucha, David G. Andersen, and Michael Kaminsky. Exploiting similarity for multi-source downloads using file handprints. In *Proc. 4th USENIX NSDI*, Cambridge, MA, April 2007.

[79] Himabindu Pucha, Michael Kaminsky, David G. Andersen, and Michael A. Kozuch. Adaptive file transfers for diverse environments. In *Proc. USENIX Annual Technical Conference*, Boston, MA, June 2008.

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

[81] S. Raoux, G. W. Burr, M. J. Breitwisch, C. T. Rettner, Y.-C. Chen, R. M. Shelby, M. Salinga, D. Krebs, S.-H. Chen, H.-L. Lung, and C. H. Lam. Phase-change random access memory: a scalable technology. *IBM J. Res. Dev.*, 52, July 2008.

[82] Sean C. Rhea, Kevin Liang, and Eric Brewer. Value-based web caching. In *Proc. Twelfth International World Wide Web Conference*, Budapest, Hungary, May 2003.

[83] RiOS 5.5 Technical Whitepaper. http://www.riverbed.com/docs/TechOverview-Riverbed-RiOS_5.5.pdf.

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

[85] Riverbed Steelhead Product Family Datasheet. http://www.riverbed.com/docs/DataSheet-Riverbed-FamilyProduct.pdf.

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

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

[88] Umar Saif, Ahsan Latif Chudhary, Shakeel Butt, and Nabeel Farooq Butt. Poor man's broadband: Peer-to-peer dialup networking. *ACM Computer Communications Review*, 37(5), 2007.

[89] Fabian Schneider, Sachin Agarwal, Tansu Alpcan, and Anja Feldmann. The new web: Characterizing Ajax traffic. In *Passive & Active Measurement (PAM)*, Cleveland, OH, April 2008.

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

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

[92] Silver Peak Systems, Inc. http://www.silver-peak.com/.

[93] F. Donelson Smith, Félix Hernández Campos, Kevin Jeffay, and David Ott. What TCP/IP protocol headers can tell us about the web. In *Proc. ACM SIGMETRICS*, Cambridge, MA, June 2001.

[94] Neil Spring and David Wetherall. A protocol-independent technique for eliminating redundant network traffic. In *Proc. ACM SIGCOMM*, Stockholm, Sweden, September 2000.

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

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

[97] Kanat Tangwongsan, Himabindu Pucha, David G. Andersen, and Michael Kaminsky. Efficient similarity estimation for systems exploiting data redundancy. In *Proc. IEEE INFOCOM*, San Diego, CA, April 2010.

[98] TCPDUMP. http://www.tcpdump.org/.

[99] Dan Teodosiu, Nikolaj Bjrner, Yuri Gurevich, Mark Manasse, and Joe Porkka. Optimizing file replication over limited-bandwidth networks using remote differential compression. Technical Report MSR-TR-2006-157, Microsoft Research, November 2006.

[100] David G. Thaler and Chinya V. Ravishankar. Using name-based mappings to increase hit rates. *IEEE/ACM Transactions on Networking*, 6(1):1–14, February 1998.

[101] Niraj Tolia, Michael Kaminsky, David G. Andersen, and Swapnil Patil. An architecture for Internet data transfer. In *Proc. 3rd USENIX NSDI*, San Jose, CA, May 2006.

[102] Universal TUN/TAP driver. http://vtun.sourceforge.net/tun/.

[103] Limin Wang, KyoungSoo Park, Ruoming Pang, Vivek S. Pai, and Larry Peterson. Reliability and security in the CoDeeN content distribution network. In *Proc. USENIX Annual Technical Conference*, Boston, MA, June 2004.

[104] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An integrated experimental environment for distributed systems and networks. In *Proc. 5th USENIX OSDI*, pages 255–270, Boston, MA, December 2002.

[105] WiMAX. http://www.wimaxforum.org/home/.

[106] Alec Wolman, Geoffrey M. Voelker, Nitin Sharma, Neal Cardwell, Anna Karlin, and Henry 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, December 1999.

[107] Youtube - broadcast yourself. http://www.youtube.com/.