AN INTEGRATED APPROACH TO IMPROVING WEB PERFORMANCE

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Lili Qiu

January 2001

AN INTEGRATED APPROACH TO

IMPROVING WEB PERFORMANCE

Lili Qiu, Ph.D.

Cornell University 2001

In the past decade, we have seen an unprecedented growth in the popularity of the World Wide Web. Such growth has created a great demand for highly efficient Web services. There are several challenges to achieving this goal.

First of all, understanding the nature of the Web workloads is critical to properly designing and provisioning Web services. However, our knowledge about the workloads of popular Web sites is still very limited.

Second, one of the primary techniques to improving Web performance is by replicating content to multiple places in the Internet. It remains an open issue how to provision such a replication infrastructure efficiently.

Third, Web transfers are short and bursty [60, 123, 86]. Such characteristics interact poorly with the dominant transport protocol – TCP. This results in low throughput and high latency.

To address these issues, we first study the dynamics of the MSNBC news site, one of the busiest Web sites in the Internet today. We analyze the dynamics of both the server content and client accesses made to the server. Our findings have significant implications for the design of efficient Web services.

Then we explore how to provision the Web replication infrastructure in detail. We develop several algorithms that automate the decision of the Web server replica

placement. Our results show that the algorithms that take workload and topology information into account can significantly out-perform the random placement scheme that is in use today.

Finally, we address the mismatch between Web transfer characteristics and TCP protocol by proposing a new technique, which we call TCP/SPAND, to effectively eliminate the slow start penalty for short data transfers, which are typical in the Web. Our results show that TCP/SPAND significantly reduces latency for short transfers without degrading the performance of connections that use standard TCP.

This thesis includes mathematical and trace-based analysis, simulation, and performance evaluation. Besides the algorithms that we have designed, our contributions include a set of paradigms for advancing the design of the Web. These paradigms include the exploitation of workload characteristics to provision Web services, and the use of shared state to optimize the transport protocol.

Biographical Sketch

Lili Qiu was born in July 1975, at Shanghai, China. Upon graduating from Nan Yang Model High School in Shanghai, she came to USA by herself at the age of eighteen. After three years of study at the University of Bridgeport, she earned Bachelor of Science degrees in Computer Science and physics, both with honors, in May 1996. Afterwards she joined the PhD program of the Cornell University Computer Science Department, where she started her journey in computer networking research.

To my parents and grandparents.

Acknowledgements

This dissertation would not have been possible without the guidance and support of many people.

First and foremost, I would like to thank my advisors, Dr. Robbert van Renesse and Professor George Varghese. I feel especially fortunate to have them as my research advisors. Robbert is a great advisor in many respects. He patiently read and carefully marked up my thesis drafts, and gave me invaluable advice and suggestions to improve the thesis. I greatly benefited from his constant support, guidance, and inspiration in my dissertation quest. George is a remarkable advisor, who grants students a lot of freedom to explore new ideas, but meanwhile interacts closely with them. He opened up a fascinating new research area for me – packet classification. I greatly appreciate and enjoy the collaboration with him.

My deep appreciation also goes to Professor Zygmunt Haas and Professor Kenneth P. Birman for serving on my special committee. Their excellent instruction and technical advice have been extremely valuable to me.

I am also grateful to Dr. Srinivansan Keshav, who introduced the computer networking field to me, and initiated my interest and passion in doing research in this area. He also kindly arranged internships for me at Bell Labs and Microsoft Research, to expose me to more practical research problems.

I am indebted to Dr. Venkata N. Padmanabhan for his excellent technical guidance and fruitful collaboration. I would like to thank him and Microsoft Research for giving me generous support, starting with a summer internship, and continuing with access to their computing facilities and unique trace data, which made this thesis possible.

I greatly benefited from the productive collaborations with Yin Zhang and Professor Geoffrey M. Voelker, which have contributed significantly to this thesis. I also got valuable comments and suggestions on various aspects of the thesis from Eva Tardos, Erich M. Nahum, Florin Baboescu, S. Ravi Kumar, Manmohan Mysore, and Brad Karp, as well as anonymous reviewers. I would like to thank them all.

Meanwhile I deeply appreciate the opportunity to interact with a number of researchers including Jonathan Rosenberg, Henning Schulzrinne, Subhash Suri, Sandeep Sibal, Satinder Singh Baveja, Li Li, and Ramin Zabih. The collaborations with them have exposed me to a number of research topics outside my dissertation, and significantly broadened my research scope.

I feel especially fortunate to have great graduate student colleagues, who have made my stay at Cornell an enjoyable and memorable experience in my life. In particular, I would like to thank Yin Zhang, S. Ravi Kumar, Mandar Mitra, Po Cho, Yanyan Xiong, Jia Wang, Zhiyuan Chen, Li Li, Zhen Xiao, Xiaoming Liu, Lidong Zhou, and Jin Huang.

I also spent one year visiting University of California, San Diego. I had a great time working with Florin Baboescu, Manmohan Mysore, and Anand Balachandran, and truly appreciate their friendship and support.

I owe a special debt of gratitude to my parents and family. Words cannot

express my gratitude to my grandparents, who had given me so much in a loving way. How I wish they could have lived to see my graduation! My parents have always given me selfless care, support, and love. Their passion for career and dedication to work have also been role models for me, and always inspired me to excel. This thesis is dedicated to them.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

With the exponential growth of the Internet, the World Wide Web has rapidly evolved into a global data dissemination system. The unprecedented success of the Web has led to a dramatic increase in Internet traffic as well as a significant degradation in user-perceived latency while accessing Web pages. These problems are bound to grow as the explosive growth of the Web continues. Improving Web performance has become one of the most important factors in determining the usability of the Web, and the well-being of the Internet as a whole.

1.1 World Wide Web

The Web is a large distributed system consisting of servers, clients, proxies, and replicas spread across the Internet. The servers are typically the originators of content while the clients[1] are the consumers of content. Proxies, when present, mediate the communication between a set of clients and a subset or all of the

[1]We use the term "client" to refer both to programs such as Web browsers and to human users.

servers. Replicas are very much like proxies, except that they act on behalf of the Web servers. For example, a content distribution network such as Akamai [3] has multiple replicas to host heavily requested documents[2] for the Web servers.

The Web contains enormous amount of information. Information in the Web is arranged as files stored on servers. A Web document is identified by a *Uniform Resource Locator (URL)*, which contains a host name and the name of a file on that host. Users interact with the Web by browsing Web pages and following hyperlinks to other pages with a Web client program called the browser. When a user click on a URL, a browser such as Netscape or Internet Explorer, queries a Domain Name Service (DNS) server for the IP address of the host, and establishes a connection to the host using its IP address. Then the clients communicate with Web servers using the *Hypertext Transfer Protocol (HTTP)* [19]. HTTP defines several *methods* that a client can specify in requests that it sends to the server. The most commonly used method is *GET*. The client issues a GET request to retrieve a resource from the server. A resource can be a static file or the output of a script executed at the server. The Web server replies by sending the requested file to the client. For performance improvement, a Web client may communicate with proxies or replicas which are closer to them than the original server, or are less loaded.

Being able to download and browse Web pages quickly is one of the principal concerns of users in the Internet. Figure 1.1 depicts the major components of the Web access latency when there are no intermediate proxies or replicas. The latency includes (i) the time taken for a DNS lookup if the browser does not have the DNS resolution, (ii) the time to set up a TCP connection, (iii) the time to send

[2]We use the terms file, document, and object synonymously.

a HTTP request, (iv) the time for the server to parse the request, (v) the time to send the HTTP response, and (vi) the time for the browser to render the page. In this thesis, we focus on reducing the network and server-side delay, such as the time spent in (ii) – (v), through Web replication and optimizing TCP performance. Several other studies, such as [38, 27], have investigated techniques to reduce or eliminate the DNS lookup time from user-perceived latency. The client's rendering time is usually negligible, but can become significant when the client machine is slow or the content to be rendered is large and complicated.

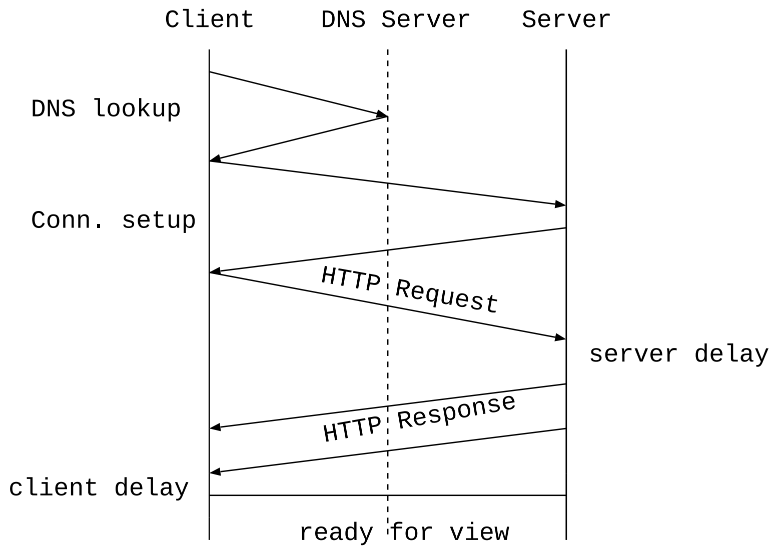


Figure 1.1: Steps in a typical Web access.

1.2 Growth of Internet and Web

The Internet, as we know it today, has evolved from the ARPANET, a research network sponsored by the U.S. Department of Defense. The original configuration in December 1969 consisted of four nodes (hosts), all located in the U.S. [101]. The ARPANET traffic was estimated a few thousand bytes per month at its inception. Since then traffic has grown exponentially, and shows no sign of stopping. In particular, the decade of the 1990s has seen a steady annual doubling of Internet traffic [37]. The only exception occurred during 1995 and 1996, when Internet traffic exhibited spectacular growth - doubling traffic every three or four months, which corresponds to an annual growth of around 1,000% [37]! By the end of 1999, the Internet traffic on U.S. long distance networks had grown to 10,000 - 16,000 Terabytes per month [37]!

The dominating status of the Web in the Internet today is underscored by recent measurements that show that Web traffic constitutes over 70% of the total on wide-area backbone links in the Internet [123].

There are many reasons behind the explosive growth in Web traffic. These reasons include: the ease of use of point-and-click interfaces; the appealing graphical content; the convenient Web-based transactions; the availability of editors and support tools for creating and publishing Web documents; an emerging trend among researchers, educational institutions, and commercial organizations to make the Web the standard mechanism for disseminating information in a timely fashion; the machine-independent nature of the languages and protocols used for constructing and exchanging Web documents; and a continuing exponential increase in the number of Internet hosts and users [108].

1.3 Challenges

The exponential growth of the World Wide Web has created a great demand for highly efficient Web services. There are a number of challenges to achieve this goal. In this section, we describe these challenges in details.

1.3.1 Understanding the Web workload

Understanding the nature of the workloads and system demands created by users of the World Wide Web is critical to properly designing and provisioning Web services.

The task of both collecting and analyzing Web workloads is a difficult endeavor. Rapid growth in Web usage, rapid changes to the software and architecture used for both serving and browsing Web documents, and the tremendous diversity of both users and content providers, all conspire to make the Web a moving target [135]. This means that characteristics observed in today's workloads may not hold true in a few years, and characteristics observed for one set of users may not hold true for another set of users. In addition, the task of workload collection is impeded by the immense size of the Internet, and by the desire for privacy on behalf of both users and content providers [135].

The study of Web workload is further complicated because the World Wide Web is not an application running at a single site. Instead it consists of a variety of different software components, such as browsers, servers, proxies, replicas, back-end databases, and other distributed applications. Thus the end-to-end performance of the Web depends on the interaction between a large number of components, which makes it very difficult to understand where the bottlenecks are in order to optimize performance for the common cases.

So far the majority of Web characterization studies have focused on data gathered at a proxy host (or by a network packet sniffer placed at a location where a proxy host might have been). Proxy-based studies are useful for many purposes: the design and evaluation of caching and prefetching algorithms, the characterization of server popularity, etc. However, proxy-based studies have their limitations because they offer only a limited perspective on the goings-on at clients and at servers. That is, a proxy is not in a position to observe *all* of the events occurring either at clients (e.g., scrolling up and down in a browser window) or at servers (e.g., the servers' communication with clients that do not connect via the same proxy).

A significant difficulty that researchers face in doing a server-based study is the very limited availability of server traces. The few pioneering studies in this area [9, 11] have had to make do with data from relatively small departmental servers, typically at universities. While they have certainly been valuable, the main limitation of these studies is that the bulk of Web traffic is served out by large commercial servers. It is unclear how well inferences drawn from the study of small departmental servers would scale to the large commercial sites in the real world.

1.3.2 Web Caching and Replication

Two primary techniques for enhancing Web performance are caching and replication. Much work has been done on caching by clients or by proxy servers that provide a shared cache to multiple clients. While clearly useful, caching alone cannot obviate the need for replication within the hosting platform. First, only static non-personalized objects can be cached. Second, caching is outside the control of the service provider, which complicates many issues (such as copyright protection, usage metering, advertisement insertion, update propagation, etc.), giving service providers an incentive to disallow caching even if objects are otherwise cachable. Overall, numerous studies have shown proxy hit ratios of only around 40% [47, 60, 55].

To address the limitations of Web caching, more and more companies are offering hosting services, that is, the service of hosting and providing access to objects belonging to third-party information providers. Successful hosting services may in the future host millions of objects on thousands of servers deployed around the globe, resulting in global information hosting systems [112].

Hosting services commonly use replication, or mirroring, to cope with load on popular Web sites and to reduce bandwidth consumption in their backbones. Since the hosting services are usually under service providers' control, the issues arising from outside control of the service providers, such as usage metering and ads insertion, in Web caching as mentioned above, are much easier to deal with.

Currently, replication (or mirroring) decisions are made by administrators, who monitor the demand for information on their sites and decide what content should be replicated and where. Making these decisions is a difficult task, and becomes even more difficult as the scale of the systems increases [112].

Furthermore, manual mirroring makes resource allocation fairly static. This implies that resource allocation has to be done for the worst-case demand scenario, or the performance would suffer. Due to large variations in user demand, network topology, and computing resources (i.e., addition or removal of servers), the cost of the worst-case approach would clearly become prohibitive. Without appropriate new technology, system administration and costs related to resource allocation may become the most important factors that limit the scale of hosting platforms [112].

1.3.3 Mismatch between the Web Application and the TCP Protocol

Web data transfers are short and bursty [60, 123, 86]. Such workloads interact poorly with TCP, the dominant transport protocol in today's Internet. TCP uses the slow start procedure [71] to probe the available network capacity both at connection start up and upon restart after an idle period. This usually requires several roundtrips. For short Web data transfers, which typically span only a few roundtrips, spending several roundtrips in probing the network state can be very inefficient. Several techniques like P-HTTP have been proposed in the past to speed up short Web transfers. These techniques help, but are not complete solutions.

1.4 Addressing the Challenges

In this thesis, we address the three challenges that we have identified in this chapter in the following ways.

1.4.1 Characterizing the Web Workload

First, we study the dynamics of the MSNBC news site, one of the busiest Web sites in the Internet today. Unlike many other efforts that have analyzed client accesses as seen by proxies, we focus on the server end. We analyze the dynamics of both the server content and client accesses made to the server. The former considers the content creation and modification process while the latter considers page popularity and locality in client accesses. Our findings have implications for the effectiveness of Web caching and prefetching, cache consistency algorithms, and optimizations such as delta-encoding [95].

1.4.2 Provisioning the Content Distribution Networks

Based on the insights obtained from the workload study, we investigate how to efficiently distribute the content to the users by leveraging the Content Distribution Networks (CDNs) infrastructure. In particular, we explore the problem of Web server replica placement in detail. We develop several placement algorithms that use workload information, such as client latency and request rates, to make informed placement decisions. We evaluate the performance of the various placement algorithms by simulating their behavior on synthetic and real network topologies using traces from large commercial and government Web servers. We also address a number of practical issues when using these algorithms online in a content distribution network, and study the sensitivity of the placement algorithms to imperfect information about client workload characteristics. As far as we know, this is the first experimental study on this subject.

Our results show that a greedy algorithm for solving the Web server replica placement problem can provide content distribution networks with performance that is close to optimal. Although the greedy algorithm depends upon estimates of client distance and load predictions, we find that it is relatively insensitive to errors in these estimates and therefore is a viable algorithm for use in the general Internet environment where workload information will always be imperfect.

1.4.3 Optimizing TCP Start-up Performance for Web Transfers

We address the mismatch between Web transfer characteristics and TCP protocol by proposing a new technique, which we call TCP/SPAND, to eliminate the slow start penalty for short data transfers. In TCP/SPAND, network performance information is shared among many co-located hosts to estimate each connection's fair share of the network resources. Based on such estimation and the file transfer size, the TCP sender determines the *optimal* initial congestion window size at connection start up and upon restart after an idle period. Instead of doing slow start, it uses a pacing scheme to smoothly send out the packets in its initial window. Once all the packets in the initial window have been paced out, the sender switches back to the behavior of standard TCP [71].

We implement TCP/SPAND in the `ns` simulator [100] and use extensive simulations to evaluate its performance. Our results show that TCP/SPAND significantly reduces latency for short transfers even in presence of multiple heavily congested bottlenecks. Meanwhile, TCP/SPAND is TCP-friendly. The significant performance improvement is not at the expense of degrading the performance of the connections using standard TCP. Therefore, deploying TCP/SPAND at Web servers can provide great performance benefit. Furthermore, our system is designed to be incrementally deployable in today's Internet. It only involves modifications at the server side. All the client side applications can be left untouched.

1.5 Thesis Outline

The remainder of the thesis is organized as follows. In Chapter 2, we present background information, and survey related work. In Chapter 3, we analyze the workload of Web servers using traces from MSNBC.

In Chapter 4, we design several algorithms to provision content distribution networks efficiently. We evaluate the performance of various placement algorithms on a variety of synthetic and realistic network topologies using real Web server traces.

In Chapter 5, we address the mismatch between Web transfers and the TCP protocol using TCP/SPAND, which optimizes the TCP startup performance for short Web transfers.

We conclude this thesis with a summary of our contributions in Chapter 6. We also point out directions for future work.

Chapter 2

Background and Related Work

In this chapter, we present background information, and survey previous work. First, we discuss Web workload characterization. Then we describe Web caching and content distribution. Finally, we review the TCP protocol, and survey a variety of transport protocol techniques in addressing Web performance problems.

2.1 Web Workload Characterization

Numerous recent studies of Web workload characterization have been performed. These studies include analyses of Web access traces from the perspective of proxies, browsers, and servers, each providing a unique perspective on the functioning of the Web. Below we overview the previous work in each of these areas.

2.1.1 Proxy-based Studies

Much of the work thus far in Web workload characterization has focused on proxies. In many instances, the goal is to evaluate the effectiveness of proxy caching. The hit ratio of proxy caches have been found to be quite low, often not much higher than

50% [47, 60]. A substantial fraction of the misses arise from *first-time* accesses to files (i.e., compulsory misses). For example, [127] has shown the first-time misses account for 20 - 40% of the requests.

Proxy logs have also been used to study the effectiveness of cooperative caching. In this context, a recent study [133] reports that the organizational membership of clients is significant in that clients belonging to the same organization are more likely to request the same documents than clients in different organizations. [8] reports that geographical, cultural and social issues have a strong influence on the workload of a proxy server.

The relative popularity of Web pages accessed via a proxy has also been studied extensively. The almost universal consensus is that page popularity follows a Zipf-like distribution where the popularity of the ith most popular file is proportional to $1/i^{\alpha}$. The value of α is typically less than or equal to 1 [59, 22, 134], which makes popularity distribution as seen by the proxy rather flat (e.g., [22] reports that it takes 25-40% of pages to draw 70% of the client accesses).

Gribble and Brewer [60] collected a trace of HTTP accesses over a modem pool at the University of California, Berkeley, over a period of 45 days and including over 8,000 clients. Their study shows there is significant heterogeneity in the client population. They also observe that clients' activity exhibits an extremely prominent diurnal cycle, and that the clients' request rate is relatively smooth at large time scales (on the order of tens of minutes, hours, or days), but increasingly bursty at smaller time scales. Moreover, they find there is a large amount of locality of reference within clients' requests. The amount of locality increases with the client population size.

Feldmann et al. [55] considers performance implications of low-level details such as cookies, aborted connections, and persistent connections between clients and proxies as well as between proxies and servers. Their study shows these details have strong impact on performance, particularly in heterogeneous bandwidth environments where network speeds between clients and proxies are significantly different than speeds between proxies and servers. In particular, they find (i) caching persistent connections at the proxy can improve latency much more than simply caching Web data; (ii) aborted connections can waste more bandwidth than that saved by caching data; (iii) "cookies" can dramatically reduce hit rates by making many documents effectively uncachable.

Proxy logs have also been used to study the rate of change and age distribution of files. For example, Duska et al [47] investigates this issue by using a set of seven proxy traces, which include a large 1996 proxy trace from DEC, four university traces, the NLANR second-level proxy cache traces, and a national trace from Korea. Their study shows that 2% to 7% of the time revalidations return unchanged objects, which indicates the inefficiency of the cache consistency control protocol.

Douglis et al. [45] uses live traces of two large corporate communities to evaluate the rate and nature of change of Web resources. They find that many resources were never modified, others were modified often, and a significant fraction were modified at least once between each traced access. The rate of change depends on many factors, particularly the content type, and the top-level domain that the resources belong to. On the other hand, the size of a resource does not appear to affect modification rates. The limitation of their study is that the number of updates may be under-estimated, since they could not detect the updates between the observed accesses. Moreover, inferring modification interval indirectly from the last-modified timestamp could also lead to missed updates.

2.1.2 Browser-based Studies

Cunha et al. collected traces of executions of NCSA Mosaic at Boston University's Computer Science department in 1995. Their results show that many characteristics of the Web use can be modeled using power-law distributions, including the distribution of document sizes, the popularity of documents as a function of size, the distribution of user requests for documents, and the number of references to documents as a function of their overall ranking in popularity [41].

In 1998, Barford et al. collected the Web client workloads from the same computing facility, and compared them with the traces captured in 1995. They found that the kinds of distributions that best described document sizes did not change between 1995 and 1998, although specific values of the distributional parameters were different. They also investigated how the observed differences in the properties of Web client requests, particularly the popularity and temporal locality properties, affected the potential for Web file caching in the network. Their results showed that for the computing facility represented by their traces between 1995 and 1998, (1) the benefits of using size-based caching policies diminished; and (2) the potential for caching requested files in the network declined [18].

2.1.3 Server-based Studies

Server-based studies are far fewer in number than proxy-based ones. A number of these have focused primarily on the network dynamics of busy Web servers [94] [15]. Others have focused on very specific aspects of the server's operation (e.g., [83] discusses the performance of reverse DNS lookups in the CNN server cluster [36]).

A few of the server-based studies have been on characterizing the access dynamics. [9] studied access logs collected in 1994 - 1995 from a set of Web servers, the busiest of which saw under 50000 accesses in a day. They showed that file popularity followed Zipf's distribution (i.e., Zipf-like with $\alpha = 1$). They also demonstrated the presence of temporal locality in file accesses. [11] studied various aspects of server behavior using data from a set of servers. They reported that 10% of the files accessed accounted for 90% of the server requests, and that 10% of the (client) domains accounted for over 75% of the server requests. However, the busiest of the servers they studied only saw a *total* of around 350,000 accesses in a day.

[88] analyzed a wide set of Web server logs collected during 1995 - 1997. Their study showed that site growth (in terms of number of hits) correlated with one of six different quantities: the number of Web users, the number of documents a user is likely to visit on a site, the number of documents on a site, the structure for charging fee for clients' accesses, the frequency with which search engines return a particular site, and the efforts of Web masters at attracting users. They also showed that, on a variety of sites, contrary to popular belief, the use of CGI did not appear to be increasing and that long latencies are not necessarily due to server loading.

More recently, Arlitt et al. [13] studied the workload characterization of the 1998 World Cup Web site. Their measurements were collected over a three month period. During this time the site received 1.35 billion requests, making this the largest Web workload analyzed to date. Their results revealed that caching at Web clients, proxies, and within the network is changing the workloads seen by Web servers. The lack of an efficient, supported, and widely adopted cache consistency mechanism is the main cause of these changes and the primary reason why Web caches are failing to significantly reduce Web server workloads during times of extreme user interest in the content on those servers.

2.1.4 Summary

In summary, there have been significant research efforts on Web workload characterization. Nevertheless rapid growth in Web usage, rapid changes to the software and architecture used for both serving and browsing Web documents, and the tremendous diversity of both users and content providers, all warrant more work in this area. In particular, the following aspects require further examination:

- Content dynamics: Content dynamics refers to the content creation and modification process at the Web servers. It is important to study because it has profound implications for the effectiveness of Web caching, in particular, the cache consistency control mechanism. Most previous studies focus on the access dynamics of Web workload, and only a few studies looked at the rate of change and age distribution. In these studies, for example [45], such information has been deduced indirectly using the last-modified timestamp in the HTTP response header, which opens up the possibility of missed updates. A more direct approach is to analyze the logs and file system activity at the Web servers.

- Access dynamics at popular Web sites: Though there have been a number of server-based studies, most studies look at small departmental Web servers, typically at universities. While they have certainly been valuable, the main limitation of these studies is that the bulk of Web traffic is served by large commercial servers. It is unclear how well inferences drawn from the study of small departmental servers scales to large commercial sites in the real world.

To address these issues, in Chapter 3, we study the dynamics of the MSNBC news site, one of the busiest Web sites in the Internet today. We analyze the dynamics of both the server content and client accesses made to the server. We see our study of the MSNBC site as complementing the existing body of literature on Web workload and traffic characterization. Both the extent of the dataset we have analyzed and the use of back-end logs to characterize the content dynamics make our study valuable.

2.2 Web Caching

Caching has been employed in many fields as an effective data access optimization to improve response time and save bandwidth, for example, in computer architecture, distributed shared memory (DSM), distributed file systems, distributed databases, and more recently the Web. Cao et al. gave a nice illustration of the differences and similarities between caching in the Web and caching in these other fields [25]. The unique challenge in the Web is to scale the caching techniques to the size of the Internet.

2.2.1 Caching Architecture

There are two widely deployed forms of caching in the Web today: browser caches and proxy caches. All popular Web browsers today support both memory and disk-based caches. Most browsers allow you to control the amount of resources given to the cache, and also the policy used by the cache to determine when to revalidate cached documents [135]. Browser caches are intended to handle Web documents reuse for a single user.

Just as browser caches handle reuse for a single user, proxy caches handle reuse for a group of users when the documents are simultaneously accessed and shared by multiple clients. The hit ratio increases as more clients share the caches.

Therefore in order to further increase the hit ratio, there has been extensive work on cooperative Web caching, which shares and coordinates cache state among multiple communicating proxy caches. Cooperative Web caching proposals include hierarchical schemes like Harvest and Squid [28, 119], hash-based schemes [74, 128], directory-based schemes [52, 91, 122] and multicast-based schemes [92, 125]. Although each of these research efforts included a performance evaluation of the protocols proposed and a discussion of scalability, only [81] presents empirical evaluations of cooperation for small populations, and none present empirical or analytical evaluations of the effectiveness of their schemes for the large client populations found in a wide-area setting [134].

[134] expands on these previous research efforts, and uses trace-based analysis to quantify the potential advantages and drawbacks of inter-proxy cooperation for small and medium-sized organizations. Their results show that cooperative caching has a significant performance benefit (i.e., increase in hit ratio) only when the population is small. For a large population (beyond 1 million users), cooperative caching is unlikely to provide a significant benefit. Therefore, cooperative caching is only necessary to adapt to proxy assignments made for political or geographical regions.

2.2.2 Cachability of Web Documents

Numerous studies have reported that cachability is the limiting factor on the effectiveness of caching. For example, [55] shows that 35 - 45% documents are uncachable due to various reasons, such as cookies, URLs with 'cgi-bin' or '?', authorization, client cache-control, and server cache-control etc. [133] reports 40% of requests are uncachable. Therefore increasing cachable documents is the key to improving the effectiveness of Web caching.

A number of research efforts attempt to address this problem. For example, Cao et al. propose an "active cache" [26], where a Web server can attach a Java application, called a "cache applet," with a document. The role of the cache applet is to be invoked when there is a cache hit to the document. When a user request hits on the cached copy and the proxy would like to service the request, the proxy must invoke the cache applet with the user request and other information as arguments. The cache applet can perform a variety of functions, e.g., logging user accesses, rotating advertising banners, checking access permissions, constructing client-specific Web pages, etc. It is shown that this scheme can lead to significant network bandwidth savings. Unfortunately, due to the significant CPU overhead of running applets, the user-perceived latency is often much worse than without caching dynamic objects.

[84] proposes a "Web server accelerator" to cache dynamic content. The accelerator resides in front of one or more Web servers to speed up user accesses. It provides an API which allows application programs to explicitly add, delete, and update cached data. Consequently, it allows dynamic Web pages to be cached as well as static ones, since applications can explicitly invalidate any page whenever it becomes obsolete.

2.2.3 Cache Consistency Control

For Web caching to be useful, cache consistency must be maintained, that is cached copies should be updated or invalidated when the originals change. There are two types of cache consistency: *weak* consistency and *strong* consistency. Weak consistency is defined as the consistency model in which a stale document might be returned to the user, and strong consistency is defined as the model in which after a write completes, no stale copy of the modified document will ever be returned to the user [25].

Most existing proxies provide only weak consistency by (i) employing a server specified lifetime of an object, which is referred to as the *time-to-live (TTL)* value, or (ii) periodically polling the server to verify that the cached data is not stale. In either case, modifications to the object before its TTL expires or between two successive polls causes the proxy to return stale data [48].

Strong consistency can be enforced either by server-based invalidation or client-driven polling. The former approach requires the server to notify proxies when the data changes. This approach substantially reduces the number of control messages exchanged between the server and the proxy, at the cost of requiring the server to maintain per-object state consisting of a list of all proxies that cache the object. The amount of state that need to be maintained can be significant especially at popular Web servers. Moreover, when a proxy is unreachable due to network failures, the server must either block on a write request until a timeout occurs, or risk violating consistency guarantees [48].

The client-driven polling requires that proxies poll the server on every read to determine if the data has changed [25]. This approach does not require any state to be maintained at the server, nor does the server ever need to block on a write request. On the other hand, frequent polling results in more control messages, higher server workload, and longer client response time.

In [25], Cao et al. compares adaptive TTL, polling-every-time and invalidation, through analysis, implementation and trace replay in a simulated environment. Their analysis shows that weak consistency saves network bandwidth mostly at the cost of returning stale documents to users. Invalidation generates a comparable amount of network traffic and server workload to adaptive TTL, and has similar average client response times, much better than polling-every-time. Therefore they recommend it as the right approach for maintaining strong cache consistency for the Web.

Duvvuri et al. studies the lease approach that balances the tradeoff between the large state space and control message overhead [48]. They use analytical models and policies for determining the optimal lease duration. Their experimental results show that their lease approach impose modest overheads, and yields a significant improvement (138-425%) over existing strong consistency mechanisms.

Krishnamurthy et al. [80] describes improving the cache consistency for the Web by using piggyback cache validation. Their idea is that when a proxy cache needs to contact a server to fetch a page, it can piggyback a set of validation requests onto the page request. The proposal in this paper suggests that the piggybacked validation requests should be generated for those pages in the cache where the TTL has expired and the page originates from the server being contacted. Using the 1996 proxy log trace from DEC and a 1996 packet trace from AT&T Research, they found that piggyback validation reduces the number of messages to the server by 17%, and reduces their cost metric, which combines the effects of latency and bandwidth, by 6%.

2.3 Web Replication

Web replication is another primary infrastructural technique to improve Web efficiency. It differs from caching in that caches store objects passively as clients' request streams traverse the caches, while replicas actively poll objects from the original servers.

Web content replication began as explicit manual mirroring of Web sites. It has since evolved into user-transparent request distribution among a static set of mirrors. The next step (especially important for large-scale Web hosting service providers) is dynamic content replication, where objects are created, deleted, or migrated among Web hosts in response to changing usage patterns [111].

Web replication is complementary to Web caching, and offers the following advantages over caching:

- Replication improves cache hit ratio for the following reasons. First, replication is under the service providers' full control. Thus issues like copyright enforcement, hit counting, and data consistency are much easier, which potentially makes more objects cachable. Second, during replication, content can be pre-pushed to the replicas before users access them. As a result, the first-time miss ratio is reduced.

- Replication provides a new type of service paradigm. Content providers have competitive demand to improve the Web performance. They cannot solely rely on proxy caching to achieve the goal, since proxy caches are not available everywhere, and are not under the content providers' control. For the clients that do not use proxy caches, their performance would still suffer. To address the need, many companies, such as Akamai [3], Digital Island [44], Exodus [51], etc., emerged to offer hosting services to the content providers. Their basic idea is to replicate content from their customers' Web sites (usually popular content providers' Web sites) to a set of geographically dispersed hosts. Replication service is valuable in that content providers now can improve their clients' performance more directly than relying on their clients to use proxy caches for performance improvement.

Two key research issues in Web replication are placement of Web replicas and mechanisms for Web request redirection.

Much of the work thus far in Web replication have concentrated on Web request redirection. The proposed methods include router-based schemes like IPv6 Anycast; server-based schemes like HTTP redirection [10], URL re-writing [50] and TCP router [43]; and client-based schemes like [54, 49], and DNS redirection [75, 23, 74].

On the other hand, less attention has been given to the placement of Web replicas. The only prior work on the placement problem that we know of is [85] by Li *et al.* They approached the placement problem based on the assumption that the underlying network topologies are trees, and modeled it as a dynamic programming problem. Although an interesting first step, this approach has a number of limitations. First, the Internet topology is not a tree, and the paper does not evaluate how well the dynamic programming algorithm based on tree-topologies works in Internet topologies. Our studies (in Chapter 4) based on real traces and topologies shows that, though the assumption of a tree topology makes it possible to obtain an optimal solution to the placement problem when the constraints are satisfied (i.e., the topology is actually a tree, and the clients can only direct requests to proxies on its path toward the Web server, but not to sibling proxies), in a more general setting it does not perform as well as the heuristics that work in general graph topologies. Moreover, its high computational complexity ($O(N^3M^2)$ for choosing M proxies among N potential sites) prevents its practical use in topologies with thousands of nodes.

2.4 Transport Protocol Optimization

Since the Web is an application running at the top of the network protocol stack, its performance is highly dependent on the transport protocol. In this section, we first review TCP, the predominant transport protocol in the Internet, which is used to carry Web and other Internet traffic. We then survey previous work on optimizing TCP for Web transfers. Finally, we review other transport protocols for improving Web performance.

2.4.1 Introduction to the Transmission Control Protocol (TCP)

TCP is currently the most dominant transport protocol in the Internet, and forms the foundation of applications such as Web browsing, E-mail, file transfer, and news distribution.

Before discussing TCP in detail, we first outline the functions that transport protocols may perform in general:

- Multiplexing: A network level address, such as an IP address, only identifies a network interface. The transport layer adds an application-specific identifier (usually called a port number) to a packet, so that the receiving end-system can hand an incoming packet to the correct application.

- Error control: The transport layer guarantees that, with very high probability, a message will reach the destination despite packet loss, corruption, and duplication. Transport-layer protocols deal with loss by transmitting lost packets; with corruption by detecting, discarding, and retransmitting corrupted packets; and with duplication by detecting and discarding duplicate packets [78].

- Flow control: This is a mechanism to enable a data source to match its transmission rate to the currently available service rate at a receiver.

- Congestion control: This is a mechanism to enable a data source to match its transmission rate to the currently available service rate in the network.

TCP is a full-fledged transport protocol that performs all of the functions listed above at the granularity of a connection. Each TCP connection performs congestion control and loss recovery independently of other connections. There is a tight coupling between its congestion control and loss recovery mechanism.

TCP Protocol Control Block (TCB)

Each TCP connection maintains state, usually in a data structure called the TCP Control Block (TCB). The TCB contains information about the connection state, its associated local process, and feedback parameters about the connection's transmission properties. Below are some important state variables in TCB:

- *sndnxt*: the sequence number of the next data byte to send

- *snduna*: the sequence number of the oldest unacknowledged data byte

- *cwnd*: the congestion window size

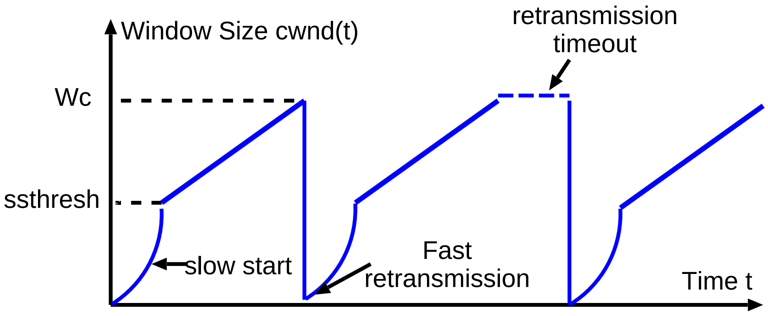
- *ssthresh*: the slow start threshold
- *srtt*: an exponentially smoothed estimate of the roundtrip time (RTT) below the sender and the receiver
- *rttvar*: the mean deviation in RTT
- *rto*: the retransmission timeout value

We will explain these variables in more details in the following discussion.

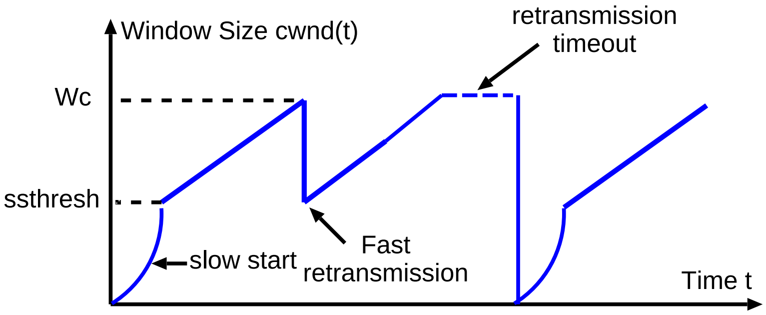
Congestion Control in TCP

The congestion control in TCP is a sliding window protocol with time-outs and retransmits. As with all sliding window protocols, the protocol has a window size that determines the amount of data that can be transmitted before an acknowledgement is required.

TCP has different flavors depending on the details of the congestion control protocol. TCP/Tahoe, TCP/Reno, and TCP/NewReno are typical TCP flavors used in today's Internet. Figure 2.1 (a) and (b) plot the congestion window (*cwnd*) evolution in TCP/Tahoe and TCP/Reno, respectively. As we can see, in both TCP/Tahoe and TCP/Reno, upon startup and restart, a source uses the *slow start algorithm* [71] to probe the available network capacity by exponentially growing its congestion window until congestion is detected or its window size reaches the receiver's advertised window. The slow start is also terminated if the congestion window grows beyond the slow start threshold (*ssthresh*). In this case, it uses the *congestion avoidance* [71] to further open up the window until a loss is detected either through retransmission timeout or duplicated ACKs.



(a) TCP/Tahoe



(b) TCP/Reno

Figure 2.1: The dynamics of the congestion window size (*cwnd*) and slow start threshold (*ssthresh*) of a TCP connection

TCP/Tahoe differs from TCP/Reno in the way that they response to losses. If a loss is detected through retransmission timeouts, in both TCP/Tahoe and TCP/Reno, the slow start threshold (*sstresh*) is set to one half of the size of the congestion window (*cwnd*) at the time the loss is detected, and the *cwnd* is reset to 1. Then the TCP sender enters the slow-start phase. If the loss is detected through duplicate ACKs, TCP recovers the loss by *fast retransmission.* During the fast retransmission, a TCP/Tahoe connection uses the same approach in adjusting the *cwnd* and *ssthresh*. In comparison, a TCP/Reno connection does the following:

1. Set *ssthresh* to half the *cwnd*.

2. Retransmit the missing segment.

3. Set *cwnd*=*sshtresh*+3.

4. Each time the same duplicated ACK arrives, increment *cwnd* by 1. Transmit a new packet, if allowed by *cwnd*.

5. If a non-duplicated ACK arrives, then set $cwnd = ssthresh$, and continue with a linear increase of the *cwnd* (Congestion-Avoidance).

The first and second actions are performed by fast retransmit. The third and forth actions correspond to *fast recovery*. The rationale behind the fast recovery is to keep the pipe flowing while waiting for the retransmitted packet's acknowledgement. Fifth, in the case of a single packet drop, the first non-duplicated ACK will open the window completely. There is no need to continue with the previous, artificially blown, *cwnd*, and we set it to *ssthresh* and continue performing congestion avoidance to find out the capacity of the line.

TCP/NewReno [67, 58] is the same as TCP/Reno, except that it improves the TCP fast recovery phase through better use of ACK information than TCP/Reno. In TCP/Reno, fast recovery would be terminated even in the case of a partial new ACK, i.e., one that does not acknowledge all data that was outstanding at the time fast retransmission was initiated. A partial new ACK implies that the next packet in the sequence was probably lost. Based on this insight, the sender using TCP/NewReno could immediately retransmit the missing packet and continue to be in the fast recovery phase, and thereby avoid the penalty of a timeout. This technique reduces the occurrence of timeouts when multiple packets are lost in the same window [101].

The success of TCP relies on the feedback mechanism, where it uses a packet loss as the indication to adapt itself through adjusting a variety of parameters, such as the congestion window size, the slow start threshold, the smoothed round-trip time, and its variance. However, for any feedback mechanism to work, a connection should last long enough. When transfer sizes are small compared to the bandwidth-delay product of the link, such as file transfers of several hundred kilobytes over satellite links or typical-sized web pages over terrestrial links, it is very likely that we have little or no feedback – loss indications. In this case, the connection's performance is primarily governed by the choice of initial parameters.

In the original TCP [71] (regardless of different flavors), the initial values of such parameters are chosen to accommodate a wide range of network conditions. They may not be optimal for a specific network scenario.

2.4.2 Previous Work on Improving TCP Startup or Restart Performance

There have been a number of proposals on improving the startup and/or restart performance of TCP connections.

Two typical examples of application level approaches are launching multiple concurrent TCP connections and P-HTTP (persistent HTTP) [104]. Using multiple concurrent connections makes TCP overly aggressive for many environments and can lead to congestive collapse in shared networks [57, 15]. P-HTTP reuses a single TCP connection for multiple Web transfers, thereby amortizing the connection setup overhead, but still pays the slow start penalty. Since the average Web document (including the inline Web objects) is only around 30KB [86], such penalty is significant enough to limit the performance benefit of P-HTTP.

T/TCP [20] bypasses the three-way handshaking by having the sender start transmitting data in the first segment sent (along with the SYN) [4]. In addition, T/TCP proposes temporal sharing of TCP control block (TCB) state, including maximum segment size (MSS), smoothed RTT, and RTT variance [20]. [21] also mentions the possibility of caching the *congestion avoidance threshold* without offering details.

Hoe [67] proposes to use the bandwidth-delay product to estimate the initial slow start threshold. They use a packet-pair [76] like scheme to estimate the network *available bandwidth*, but it does not work well for FIFO networks.

Allman *et al.* [5] propose to increase the initial window (and optionally the restart window) to roughly 4K bytes. There have been various studies in support of [5] such as [7] and [117]. However, since they use a fixed initial window for all connections, the value has to be conservative, and the improvement is still inadequate in situations where the bandwidth-delay product is much larger.

TCP control block interdependence [124] proposes temporal reusing TCP state, including carrying over congestion window information from one connection to the next. Similar to [124], Fast Start [101, 103] reuses the congestion window size, the slow start threshold, the smoothed round-trip time, and its variance. By taking advantage of temporal locality, both approaches can lead to a significant performance gain. On the other hand, as demonstrated in Section 5.1, the optimal initial congestion window size depends on both the *network state* and the *transfer size*. Directly reusing previous parameters is not optimal. In addition, when network condition changes, such reusing can be overly aggressive. Fast Start addresses the issue by resorting to router support. This solution is difficult to deploy in today's Internet. Moreover, in both approaches information sharing is limited to within a single host (although the possibility for inter-host sharing is mentioned in [101]).

Rate-Based Pacing (RBP) [129] also reuses the value of *cwnd* from before the idle period, but it smoothly paces out packets rather than burst them out back-to-back. In order to take into account of the fact that the network condition may change after the idle period, they suggest pacing out the packets at the rate reduced by a constant factor. However it is very tricky to find an appropriate constant factor as a result of the length of the idle period. Moreover directly reusing the previous *cwnd* is not optimal.

In summary, TCP start-up performance has received considerable attention. Many proposals help, but are not complete solutions.

2.4.3 Other Transport Protocols

In this section, we briefly survey other transport protocols in addressing the Web performance problem.

Packet-Pair Bandwidth Probing

Packet-pair [77] is a technique for a host to quickly discover its fair share of the network bandwidth. In this approach, the sender transmits a pair of packets back-to-back and measures the time gap between the corresponding ACKs. If the router at the bottleneck link uses the fair queuing scheduling algorithm [42], the interval between the ACKs is inversely proportional to the connection's service rate at the bottleneck.

Packet-pair probing could be used to initialize the state in TCP. However it has the following limitations that impede its use. First, it works only if routers in the network use fair queuing. But most routers in today's Internet implement FIFO scheduling due to the high cost of maintaining per-flow state required in fair queuing. Secondly, packet-pair takes at least one roundtrip time for bandwidth probing, which may be costly for short transfers.

Hoe [67] proposes using packet-pair to determine an appropriate initial value for the slow start threshold in TCP. But it does not work well with FIFO networks, and still pays slow start overhead.

WebTP

The WebTP protocol is a transport protocol designed specifically for the Web [62]. It is receiver-driven, and uses Application Level Framing [34]. One of the major drawbacks is that it requires modification to Web browsers, as well as Web servers, which is impractical. Furthermore, a receiver-driven approach is vulnerable to malicious receivers launching denial-of-service attacks.

2.5 Summary

In this chapter, we presented the background information relevant to our work, and surveyed the previous work in this area. In the next chapter, we present our work on the workload characterization of the MSNBC Web site.

Chapter 3

Analysis of Content and Access Dynamics of a Busy Web Server

In this chapter, we study the dynamics of the MSNBC news site [96]. It is a large commercial news site in the same category as CNN [36] and ABCNews [1]. It is consistently ranked among the busiest sites in the Web [90]. The trace data we obtained was of two kinds: (a) *content logs*, which record file creation and modification events, and also include copies of successive versions of a subset of the files, and (b) *access logs*, which record client accesses to the HTML content (but not to the inline images). Besides traces from several "normal" days, our data set includes traces from a couple of days that saw significant flash crowds because of a hot news story. The combination of content logs and access logs enabled us to study both the *back-end* (i.e., content dynamics) and the *front-end* (i.e., access dynamics) of the MSNBC site. Note that MSNBC did not make use of content distribution network (such as Akamai) during the period under study. Therefore the requests recorded in the accesses logs are sufficient to study access dynamics of the MSNBC site.

A detailed discussion of the results appears later in the chapter, but here are some of our key findings: (a) file modification, although more frequent than file creation, often tends to change little in the file being modified, (b) a small subset of the files tends to get modified repeatedly, (c) file popularity follows a Zipf-like distribution with a parameter α that is much larger than what was reported in previous, proxy-based studies, and (d) there is significant temporal stability in file popularity but not much stability in the domains from which clients request the popular content. Our findings have implications for the effectiveness of Web caching and prefetching, cache consistency algorithms, and optimizations such as delta-encoding [95]. We discuss these in more detail later in the chapter.

The rest of this chapter is organized as follows. We present a discussion of the architecture of the MSNBC site, our trace collection methodology, and the traces themselves in Section 3.1. Then in Section 3.2 and 3.3, we present a detailed analysis of the content logs and the access logs, respectively. In Section 3.4, we summarize our key results and discuss the broader implications of our findings.

3.1 Experimental Setup and Methodology

In this section, we briefly describe the essential aspects of the MSNBC server site, and discuss the trace data that we gathered and processed.

3.1.1 Server Site Architecture

The MSNBC server site comprises a cluster of over 40 server nodes, each running the Microsoft Internet Information Server (IIS) [93]. These server nodes are grouped together into sub-clusters containing approximately 6 nodes each. Load balancing is done at two levels. First, each sub-cluster is assigned a *virtual* IP (VIP) addresses, and DNS round-robin cycles through the 6 VIP addresses for the entire MSNBC site. Second, within each sub-cluster that shares a VIP address, the Windows Load Balancing Service (WLBS) [93] is used to spread load evenly across the nodes. The main point to take away is that at different times, a particular client's request may be served by any of the 40 nodes.

3.1.2 Server Access Logs

Each server node maintains a standard HTTP access log that records several pieces of information for each client access: a timestamp, the client's IP address, the URL accessed, the response size, the server status code, etc. For administrative reasons, the server site operator chose to turn off logging for image accesses. So the logs only record accesses to HTML content. While the absence of image access logs is a limitation of our data set, we do not believe it interferes with our study in a significant way since we are still able to analyze access dynamics at the granularity of Web *pages* (i.e., the HTML content).

Table 3.1 summarizes the overall statistics of the server access logs we used in our study. For our analysis, we picked traces from several different periods, each spanning a few consecutive days. In some periods, we had only an hour's worth of traces per day, while in others we had traces from the entire day. The traces from 12/17/98 and 12/18/98 [1] were especially interesting, because these corresponded to a "hot" news event, namely the launching of *Operation Desert Fox* by the US military against Iraq. All the other traces in Table 3.1 are from 1999.

[1]Throughout this thesis, dates appear in the format *month/day/year.*

Table 3.1: Overall trace statistics.

	12/17/98 - 12/18/98	8/1 - 8/5	8/3 - 8/5	9/27 - 10/1	10/7 - 10/11	10/14 - 10/18
Period	9 AM - 12 AM	10 - 11 AM	9 AM - 12 AM	all day	all day	all day
% Total server logs used	100	100	100	30	22.50	22.50
# HTTP requests	10,413,866	7,061,572	14,183,820	38,191,388	28,102,751	30,000,981
# Objects	34,443	30,199	35,359	57,053	60,323	52,429
# Clients	253,484	440,151	656,449	1,948,609	1,831,027	1,938,437
% success for reverse DNS	58.587	-	76.227	78.967	78.344	-
# Domains	41,025	-	75,569	117,615	396,528	-
% GET requests	99.818	99.050	99.065	99.065	99.008	99.059
% POST requests	0.088	0.448	0.464	0.474	0.512	0.475
% HEAD requests	0.082	0.406	0.392	0.327	0.350	0.336
% Other requests	0.012	0.096	0.079	0.134	0.130	0.130
% OK responses	58.084	55.913	57.104	56.195	55.088	54.744
% Moved temporarily	4.554	15.017	15.267	17.647	16.047	18.443
% Not modified responses	36.946	27.529	26.231	23.812	26.517	24.501
% Unsuccessful responses	0.416	1.541	1.398	2.346	2.348	2.312

The MSNBC server site saw, on average, over 25 million client accesses each day to its HTML content alone (image hits were in addition to this). We used all 40 server nodes when analyzing the 1-hour or the 3-hour long traces. Due to disk and memory limitations, we only used logs from 9 or 12 (i.e., 22.5–30%) of the server nodes out of the cluster of 40 when analyzing the logs from a whole day period. Since requests are randomly dispatched to the server nodes, considering only a subset of the server nodes is unlikely to bias or otherwise impact the results of our analysis significantly. Moreover, most our results based on the partial set of server nodes are consistent with those based on all the server nodes.

In some of our analyses, we clustered clients together into *domains*, which we defined to be all but the hostname part of the clients' DNS names (e.g., the domain for foo.bar.com is bar.com). We determined the DNS name of a host via a reverse DNS lookup on its IP address. We had a fairly high success rate - typically over 70% - as reported in Table 3.1. For the analyses that involved domain information, we ignored clients for which the reverse DNS lookup failed.

3.1.3 Content Creation and Modification Logs

The back-end of the MSNBC site uses the Microsoft Content Replication System (CRS) [93] to replicate content from a staging server to each of the 40 server nodes. Each replication event is logged, specifying the time of replication and the file being replicated together with its size. However, all that a CRS log entry says is that a file was replicated, which by itself does not enable us to determine whether a new file was created or an existing one was updated. We disambiguate between file creation and modification by using several days' worth of CRS logs to prime our list of files that already exist, and thereafter (once the spike in the number of file "creation" events has subsided) treating CRS replication events for files not seen before as file creations[2]. The CRS system did not log file deletions, although it could have. The CRS logs we analyzed corresponded to the 4-week period from 10/1/99 through 10/28/99.

3.1.4 Content Logs

For a small fraction of the content hosted by MSNBC, we were able to obtain a log of the HTML content itself, that is, successive versions of the files and when they were modified. A new version of the file was logged on the hour if the file had been modified (at least once) in the past hour. The subset of files logged in this manner was determined by the server site operator[3]. The content log, although limited in scope, enables us to gain some insight into the evolution of files as they undergo modification.

3.1.5 Proxy Logs

In some of our analyses (Section 3.3.4), we compare the access characteristics of the MSNBC server with that of a busy caching proxy. We obtained logs from a proxy cache that serves the Microsoft corporate campus with a population of over 50000 client hosts. The logs were gathered over a 2-day period - 10/6/99 and 10/7/99.

[2]As a validation of this heuristic, we confirmed that the number of file creation events beyond the priming period does not diminish with time, as would have happened had the file creation events been "bogus".

[3]One of the reasons the site operator generated the content log was to feed it into various search engines for re-indexing the corresponding pages.

3.2 Server Content Dynamics

In this section, we analyze the dynamics of file creation and modification. The content dynamics is important to study because it has profound implications for the effectiveness of Web caching, in particular, cache consistency control mechanism.

3.2.1 File Creation and Modification Processes

We studied the dynamics of file creation and modification using information derived from the CRS logs. We consider the time of replication of a file by CRS to be its creation/modification time because it is only after (the new version of) a file is replicated onto the server nodes that it becomes available to clients. There is a diurnal cycle as well as a weekly cycle to the file creation and modification processes. The diurnal cycle exists despite the global accessibility of the MSNBC Web site because the bulk of the accesses to the site come from North America, that is, from time zones that are clustered together.

In Table 3.2, we tabulate a more detailed breakdown of the event counts during the one-week period from 00:00 hours on 10/9/99 through 23:59 hours on 10/15/99. We note there are nearly four times as many file modification events as creation events during the course of the week. A closer examination of the modification events reveals that they tend to be concentrated on a small number of files. On average, there were around 10 modification events per file (only considering files that were modified at least once during the week).

The large numbers of creation and modification events have broad implications. The creation of new files poses a challenge to latency/bandwidth saving schemes such as prefetching [105, 53] or server-initiated "push" [125]. These

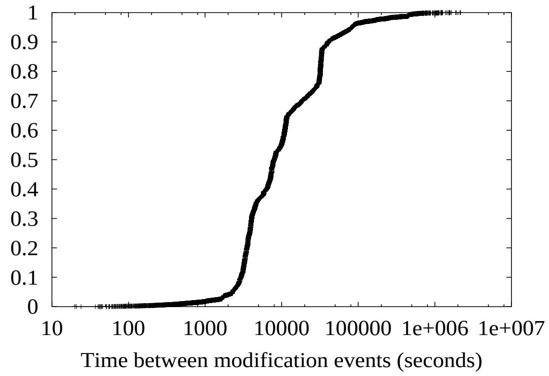


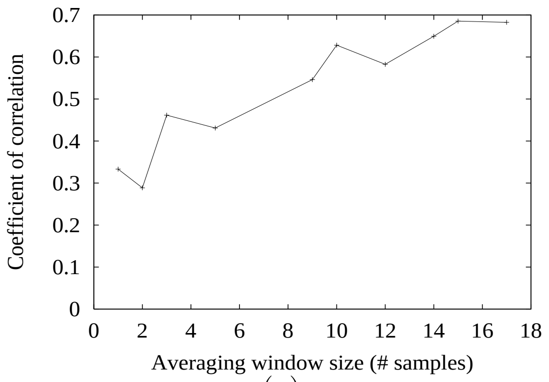
Figure 3.1: CDF of the time interval between successive modifications of a file (conditioned on the file being modified).

Table 3.2: Count of various events during the week from 10/9/99 through 10/15/99.

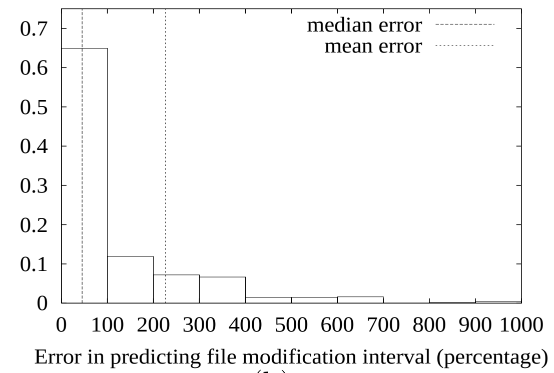
Creation	Modification	Unique Files Modified	GIF/JPEG Modification
6007	23343	2453	287

schemes depend on the past history of accesses to the file. But for a newly created file, there exists no such history. The large number of modification events suggests that it may be worthwhile deploying efficient protocol mechanisms for the validation/invalidation of files cached at proxies and/or clients (e.g., [39, 136]).

Table 3.2 also reveals that around 1% of the modification events corresponded to GIF/JPEG image files. Intuitively, we would not expect images to get modified very much; instead we would expect new images to be assigned new names. On closer examination, we discovered that the images being modified were almost exclusively maps, primarily weather maps but also maps of other kinds (e.g., a weekly "health" map indicating the current status of disease outbreak in the country). Interestingly, however, the cachability of images is lower than that of HTML due to server policy [133]. For example, advertisements mostly consist of images, and are often not cachable.



(a)



(b)

Figure 3.2: (a) The coefficient of correlation between the modification interval for a file and that predicted based on the average of past samples computed over various averaging window sizes. (b) A histogram of the percentage error in the prediction when the averaging window size is 16 samples. The median error is 45% and the mean error is 226%.

3.2.2 Distribution of Modification Intervals

Next, we turn to Figure 3.1, which shows the cumulative distribution function (CDF) of the time duration between two successive modifications (i.e., the *modification interval*) of a file. (Note that we only considered files that were modified during the 4-week period of our CRS logs. In other words, the CDF is conditioned on the file being modified during the 4-week period.) The CDF exhibits two distinct knees. The first is around the 5% level and occurs at a modification interval of around 3000 seconds (about 1 hour). The second is around the 95% level and occurs at a modification interval of around 80000 seconds (approximately 1 day). Both of these observations are in agreement with our intuition. By default, the CRS replication process is scheduled to run approximately once an hour. More frequent updates happen only when there are "hot" news events, which is typically an infrequent occurrence. Hence there are only a small number of instances of modification happening within an hour of the previous one. At the other end of the spectrum, a day seems like a natural "upper bound" for the update period. Of course, it is certainly possible for files to be modified on longer timescales such as weekly, monthly, etc. (or even aperiodically), but this is likely to be an infrequent occurrence.

3.2.3 Implications of Modification History

We now turn to examining the relationship between successive modification intervals of a file. We are interested in determining whether the modification dynamics of a file in the past is a good predictor of the future. This finding would have significant implications for Web cache consistency mechanisms such as adaptive TTL [63].

We estimate the new modification interval for a file to be the mean of the past w samples of the modification interval where w is the size of the window used for averaging. Figure 3.2(a) shows the coefficient of correlation between the estimate and the true modification interval as a function of the averaging window size. We observed that the coefficient of correlation starts off low (around 0.25) when few samples are used to derive the estimate. As more samples are used, the estimate gets better. However, beyond 15 samples the coefficient of correlation tapers off at just under 0.7. Thus the modification interval for a file correlates moderately well with an estimate based on a sufficient number of past samples. However, inclusion of more samples does not help beyond a point.

In Figure 3.2(b) we take a closer look at the accuracy of an estimate of the modification interval of a file based on past samples. The figure plots a histogram of the error in the estimate (in percentage terms). The error lies within 100% (i.e., a factor of 2) and 400% (i.e., a factor of 5) in 65% and 90%, respectively, of all instances. The median error is 45%. This suggests that while accurate prediction of future file modification behavior may not be feasible, rough estimation is possible. For instance, if the average modification interval for a file based on past samples is 10 minutes, then we can be quite confident that the next modification of the file will happen say within an hour. While this is not a tight estimate, it would still be useful to know that a file is likely to be modified at the timescale of hours rather than days.

Our results suggest that past file modification behavior, when averaged over a sufficient number of samples, can yield a useful TTL setting (e.g., one hour versus one day) in the context of an adaptive TTL-based cache consistency scheme (e.g., [63]). However, the limited accuracy of the estimate means that there is a need for an alternative consistency mechanism (e.g., callback-based invalidation) as backup.

3.2.4 Extent of Change upon File Modification

We now turn to the question of how much a file changes upon modification. To answer this question, we use the content logs which, as described in Section 3.1.4, were available for a small subset of the HTML content. For two successive versions, v_1 and v_2, of a file, we compute the delta (i.e., difference) between them, $vdelta(v_1, v_2)$, using the *vdelta* algorithm [68]. Previous studies (e.g., [95]) have shown that the *vdelta* produces a more compact delta than most other algorithms. We quantified the extent of change between the two versions of the file as $\frac{|vdelta(v_1,v_2)|}{\frac{|v_1|+|v_2|}{2}}$ where $|f|$ denotes the size of f. In other words, we quantified the size of the *vdelta* as a fraction of the average size of the pre-modification and post-modification versions of the file. The cumulative distribution function (CDF) of this quantity, expressed as a percentage, is shown in Figure 3.3. We observe that the extent of change tends to be small. In 77% of instances, the size of the *vdelta* is less than 1% that of the file undergoing modification. It is less than 10% of the file size in 96% of instances.

On closer examination, we found (at least) a couple of common modes of (minor) textual change that may explain these results: (a) a date/time string contained within the TITLE HTML tag is updated from time to time, and (b) the links to related pages are updated while leaving the bulk of the page unchanged. Moreover, when major modifications happen (e.g., when a new and fairly different version of a news story is generated), a new file (with a new name) is created rather than an existing file being modified.[4]

[4]Anecdotal evidence suggests that other news sites such as CNN do the same.

Our results suggest that techniques such as delta-encoding [95] would be very useful, especially in the context of slow links (e.g., dialup links).

3.2.5 Summary of Content Dynamics

In summary, our study of server content dynamics reveals that (i) files are modified and created frequently; (ii) the amount of modification is relatively small; and (iii) past modification behavior, with sufficient averaging, is a rough predictor of future modification behavior. These server dynamics have significant implications for the design of efficient protocols for the Web. In particular, it suggests that the potential benefit of an efficient cache consistency mechanism is large and that an adaptive TTL-based scheme is likely to be useful.

3.3 Server Access Dynamics

In order to further explore the impact of the server content dynamics on the Web traffic and to better understand the effectiveness of Web caching and prefetching, in this section we study the access dynamics seen by the MSNBC server site. Access dynamics is crucial to the effectiveness of Web caching. In particular, insights into the following issues may have a significant bearing on how we should improve the efficiency of the future Web:

- Relationship between server content dynamics and access dynamics, and potential benefit of cache consistency mechanisms
- Stability of user access patterns
- Significance of a user's domain membership

- The applicability of Zipf's law to Web requests at a server compared to the previous studies of requests seen at proxies

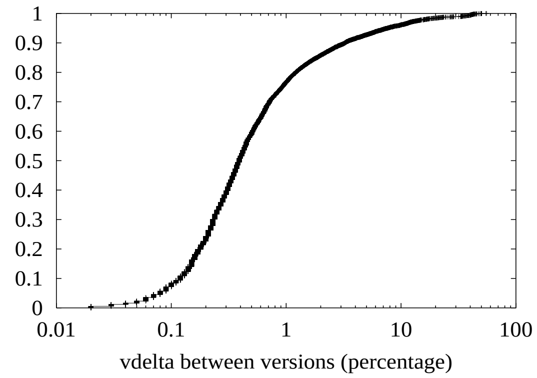


Figure 3.3: The CDF of the extent of change between succession versions of files. The extent of change is quantified as the ratio between the size of the *vdelta* difference between two versions of a file and the (average) size of the two versions. The ratio is expressed as a percentage.

As shown in Table 3.1, over 99% of the requests employ the GET method. Moreover, among all the requests, the ones with HTTP response "OK" (status code 200) account for 55%. Around 15% to 18.5% of the requests have response "moved temporarily" (status code 302), and around 23% to 37% have response "not modified" (status code 304). The latter implies that the potential benefit of an efficient Web objects consistency protocol could be large. Unless otherwise specified, in our analysis of the access dynamics, we do not distinguish among different HTTP methods (i.e., GET, HEAD, and POST) and different status code. Instead we treat them all equally as Web accesses.

3.3.1 Relationship between Server Content Dynamics and Access Dynamics

In this section, we analyze the relation between server content dynamics and access dynamics.

Correlation between age and popularity

First, we examine the correlation between document age (i.e., the time elapsed since its creation) and popularity. Our results are shown in Figure 3.4, where the x-axis denotes the time elapsed since document creation and the y-axis denotes the document ID sorted in decreasing order of popularity (i.e., document #0 is the most popular one). A dot at (x, y) in the figure means a document y is accessed at time x.

It is evident from the graphs that most documents receive a lot more accesses soon after their creation than afterwards. This happens because most news page are accessed when they are fresh. On the other hand, there are a number of documents (the ones denoted with a small document ID) that remain hot for the entire five-day period under study. These are typically index pages, such as the default front page for the MSNBC site.

Classification of accesses based on file creation and modification events

Previous research [127] has shown that up to 40% of accesses are to the objects that have not been accessed before by clients in the same domain. In a caching context, these lead to *first-time* (i.e., compulsory) misses. It is very useful to understand what comprises these first-time accesses/misses. Using both access logs and modification logs, we have found that most first-time accesses (which we

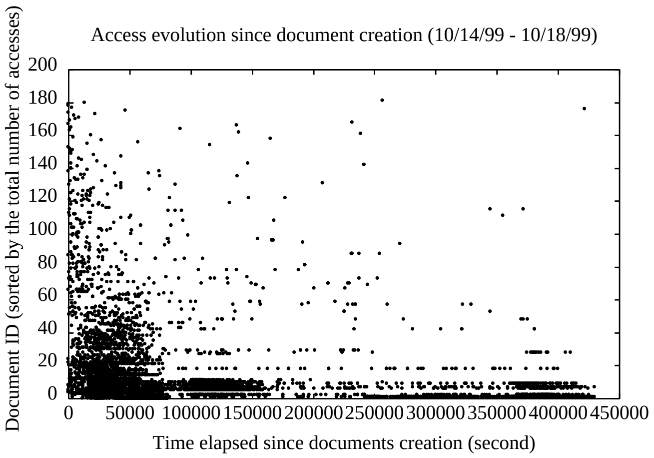


Figure 3.4: Distribution of accesses to documents since their time of creation.

determined on a per-domain basis) are to old objects that were created at least a day ago. This is evident from Table 3.3 (columns 2 and 3). We believe such accesses are very hard to predict. This is because objects that have not been accessed by a certain domain for several days following their creation are considered unpopular. Accesses to unpopular documents are hard to predict. Moreover, because these documents are unpopular, it is not cost-effective to prefetch them in advance.

We also classify repeated accesses (i.e., accesses to previously requested documents by the domains) according to their modification history. Specifically, we divide the accesses into two groups: accesses to objects that are modified on the day of access, and those that are not. The former requires a consistency control mechanism to ensure users always get the latest versions, while caching alone is sufficient for the latter type of accesses. As shown in Table 3.3 (columns 4 and 5), over half of the repeated accesses are to the modified objects, but the total number of modifications is several orders of magnitude smaller than the total number of accesses. This implies server-based invalidation can be a very efficient means of ensuring strong consistency, especially compared to client polling on every read [25, 136].

Table 3.3: Breakdown the Web accesses according to file modification/creation history.

Date	First & New	First & Old	Repeated & Mod		Repeated & Unmod
			# Mods	# Accesses	
10/8/99	72362	240119	7937	1317246	1014821
10/9/99	14652	96167	6051	697338	576113
10/10/99	15156	99248	6107	758626	610435
10/11/99	47619	206743	7324	1163424	919085

3.3.2 Temporal Stability of User Access

In this section, we analyze the temporal stability of Web accesses with the following questions in mind:

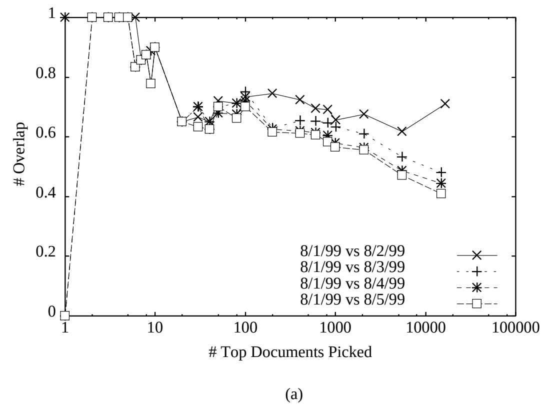
- How much does the popularity of Web pages change with time? That is, do popular Web pages on one day remain popular on the subsequent days?
- For each Web page, how much does the set of domains interested in it vary from one day to the next?

Answers to these questions are critical for designing sensible prefetching or server-initiated push algorithms. Any prefetching algorithm based on past history relies on a certain degree of stability, both in popularity ranking and in the interest group (i.e., the set of domains requesting the pages). Our trace analysis sheds light on how well such reactive prefetching algorithms might perform in practice.

Stability of Web Page Popularity Ranking

We study the stability of Web page ranking as follows. We consider Web access logs from several consecutive days. For each day, we pick the n most popular documents. We then compute the overlap in the most popular pages selected in one day to those selected in the next day. Our results are illustrated in Figure 3.5.

On the x-axis is the number of most popular documents picked (e.g., $x = 10$ means we pick the 10 most popular documents), and on the y-axis is the percentage of overlap. Figure 3.5(a) plots the ranking stability in a week period. We make the following observations. First, the overlap is mostly over 60%, which essentially means many documents are hot on both days. Second, for the several consecutive days period, the overlap is mostly the same. For example, the amount of overlap between 8/1/99 vs. 8/2/99 is quite close to that between 8/1/99 vs. 8/5/99. This implies that the Web accesses within a short time-frame (such as a week) are equally good for predicting which Web pages will be among the most popular in the near future. That is, last week's trace is almost as useful as yesterday's trace for predicting Web page ranking. This happens because many of the very popular pages are *index* pages (such as the default front page for the MSNBC site) that contain pointers to other pages. Third, the ranking stability tends to decrease as the number of selected documents increases. This indicates that very "hot" documents are more likely to remain hot than moderately hot documents. We observe similar results in the traces from other periods as well.



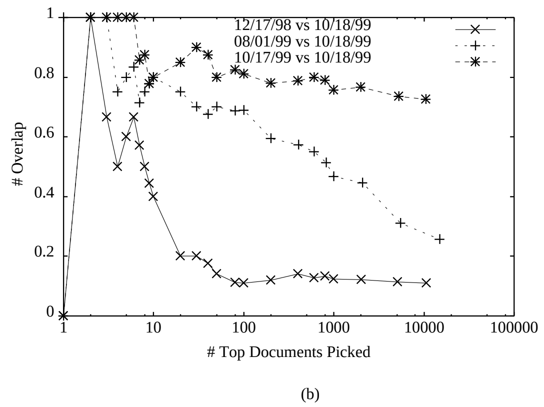


Figure 3.5: Stability of document popularity in both a short and a long period.

We also study the ranking stability in a larger time window considering the overlap between the two days that are more widely separated. Our results are shown in Figure 3.5(b). As we would expect, the overlap decreases as the time interval gets longer. For example, the overlap between 12/17/98 and 10/18/99, which are 10 months apart, is considerably smaller, mostly below 20%. On the other hand, although the overlap for the two months separation (8/1/99 and 10/18/99) is lower than with a one-day separation, it is still quite significant. For the top 100 documents, the overlap is above 60%. Compared to the one day separation, the overlap in the two month separation decreases much faster as the number of documents selected increases.

We further explore this issue by breaking down the overlap and disjoint regions into four groups (assuming Day 1 is prior to Day 2):

- *Common & unmodified*: documents that are popular on both days and have not been modified since Day 1
- *Common & modified*: documents that are popular on both days, and have been modified since Day 1
- *Different & old*: documents that are popular on only one of the days, but exist on both days
- *Different & new*: documents that are created after Day 1, and are popular only on Day 2

Our results are shown in Figure 3.6. We observe that many of the popular files in common between the two days are unmodified throughout the period (i.e., starting from day 1 through day 2). This implies that caching is potentially able to reduce Web traffic significantly. However, there exists a significant discrepancy between the number of files that are potentially cachable (i.e., not modified) and the number of files that are actually allowed to be cached by servers [133]. Bridging this gap will help reduce Internet traffic substantially as well as improve client latency.

The disjoint region in the set of popular files on the two days is mostly not due to the creation of new files, since most of the disjoint region consists of the accesses to files that are in existence on both days.

To summarize, in this section we studied the ranking stability of Web pages, and found that the stability is reasonably high on the scale of days. The ranking tends to change only gradually over time. We also examined what contributes to the overlap and disjoint regions of popular documents between a pair of days. Our results show the disjoint region mostly consists of old pages, not newly created pages. In the overlap region, the number of modified documents exceeds the number of unmodified documents, though both counts are of the same order of magnitude.

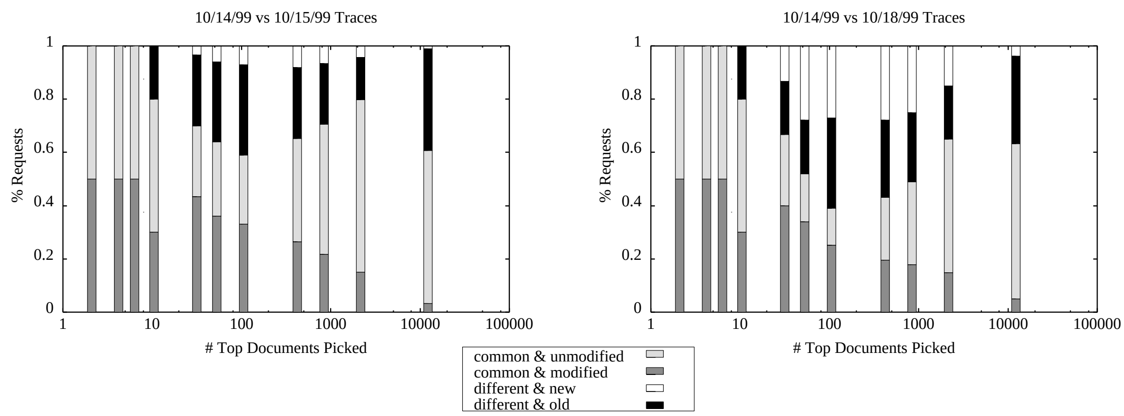


Figure 3.6: Stability of document popularity.

Stability of Interest Group

We now consider how the interest group for each Web page changes over time. Our approach is as follows: For every Web page that receives over 100 accesses on the first of the pair of days in our study, we find the set of domains which accesses the page on each day, and determine the extent of overlap between these sets. As mentioned in Section 3.1.2, we ignore requests from clients for which the reverse DNS lookup fails. Since the percentage of failure is reasonably low (Table 3.1), it should not make a significant difference in our results.

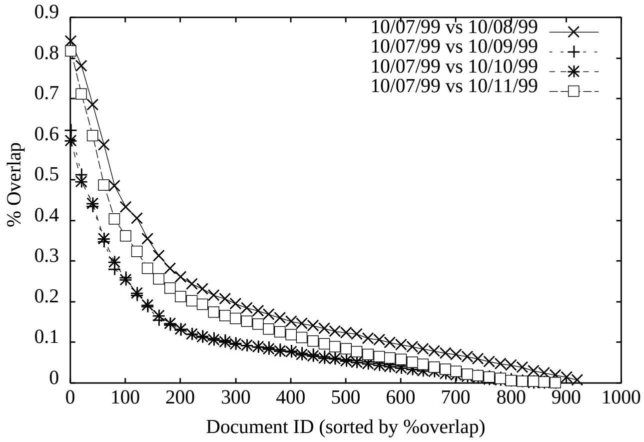


Figure 3.7: Stability of interest group.

Figure 3.7 shows the extent of overlap (as a percentage) for several pairs of days we studied. As we can see, the overlap is not very large. Only a few documents have over half of the domains making requests on both days. We observe similar results during other periods of traces as well. One explanation for this can be that the interest groups for the documents may not stabilize within a day, possibly because our definition of domains is too fine-grained. Another explanation could be that domain-level proxy caching can reduce the likelihood of multiple requests for a Web page emanating from a domain. As part of our future work, we plan to investigate why there is a large variation in the set of domains that request a Web page, from one day to another.

3.3.3 Spatial Locality

In this section, we investigate whether the domain membership is significant, that is, whether clients belonging to the same domain are more likely to share requests than clients picked at random. This kind of spatial locality has obvious implications for performance, particularly with respect to the effectiveness of proxy caching.

Our approach, based on that of [133], is to compute the degree of local sharing (i.e., intra-domain sharing) and non-local sharing (i.e., inter-domain sharing) under the following two situations: (i) when clients are assigned to domains based on their DNS names (*true* assignment), and (ii) when clients are assigned to domains randomly, while preserving the size of each domain as before (*random* assignment). A major distinction between our study and [133] is that our domains correspond to (often) large and diverse DNS level-two domains whereas [133] defines "domains" to be the rather small and much less diverse departments within a single university.

The top two graphs in Figure 3.8 show the intra-domain and inter-domain sharing on 12/17/98, and the lower two graphs show the results for 10/7/99. In both cases, the inter-domain sharing with random assignment is comparable to that with true assignment, as we would expect. In contrast, intra-domain sharing with true assignment is noticeably higher than with random assignment for 10/7/99. This is also observed in the traces of many other periods. On the other hand, the intra-domain sharing on 12/17/98 (the day of Operation Desert Fox) using true assignment is comparable to using random assignment.

From these results, we conclude the following. In most cases domain membership is significant, that is, clients belonging to the same domain are more likely to share requests than clients picked at random. This is in agreement with the findings reported in [133] despite our domains being much larger and more diverse. However, when there is a "hot" event, the global interest can become so dominant that even clients picked at random tend to share many requests, thereby diminishing the significance of domain membership.

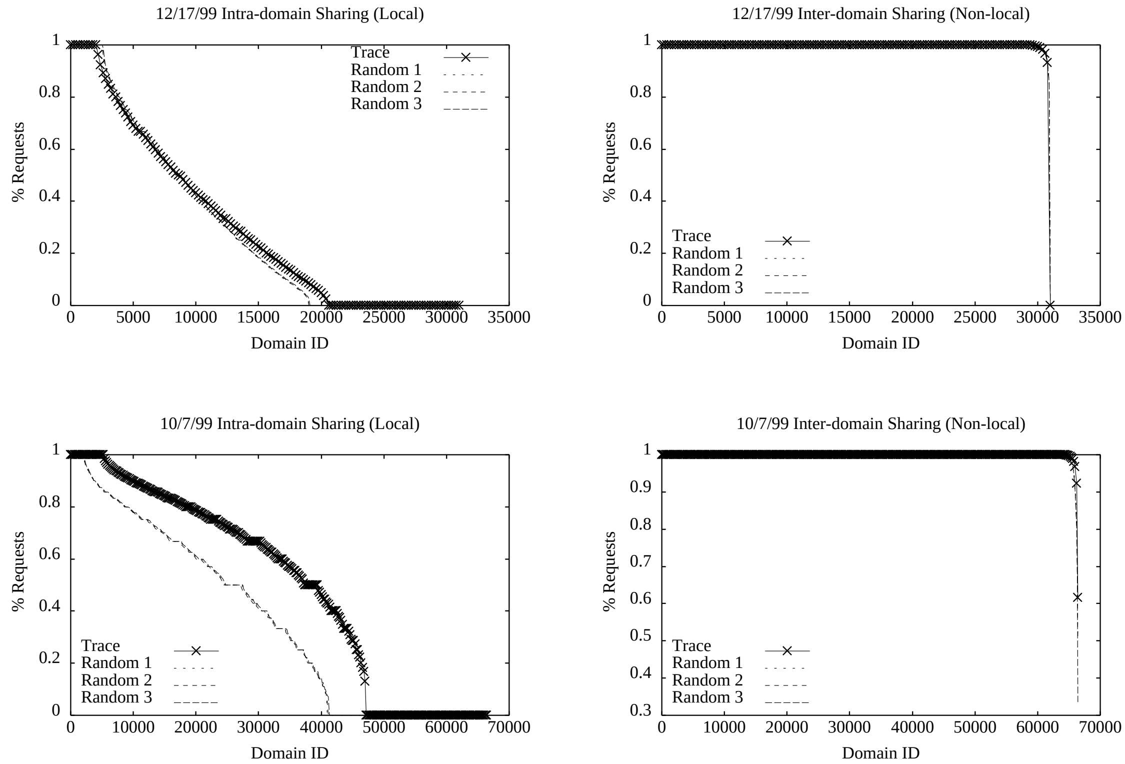


Figure 3.8: Comparison between the intra-domain and inter-domain sharing in the real traces with four random client to domain assignments.

3.3.4 The Applicability of Zipf's Law to Web Requests

Several prior studies have investigated the applicability of Zipf's law to Web accesses. The majority of studies have been based on proxy traces. These studies have concluded that the popularity of Web requests follows a Zipf-like distribution, C/i^{α}, with α close to or smaller than 1 [59, 41, 98, 22]. There have been a few studies based on traces from relatively small servers. For instance, [9] showed that the distribution of accesses seen by a Web server follows Zipf's law (i.e., $\alpha = 1$).

We investigate this issue further by studying access logs from both the server and the proxy. We plot the number of document accesses versus document ranking on log-log scale. Figure 3.9 and Figure 3.10 show the plots for the server traces and proxy traces, respectively.

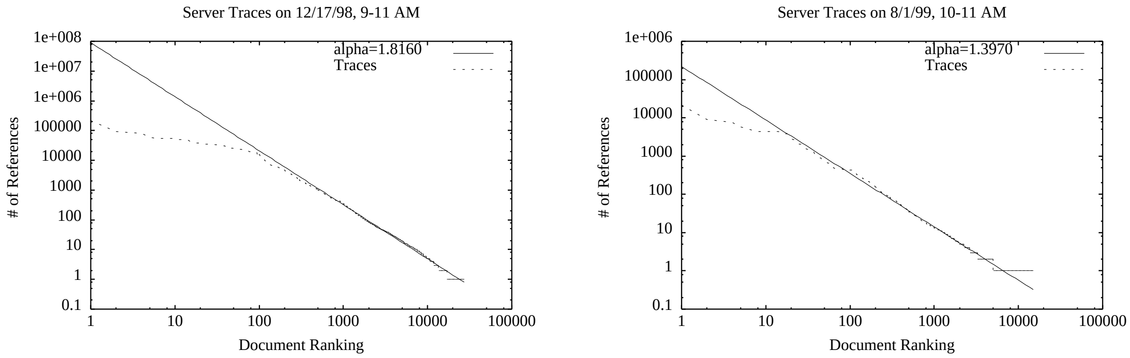


Figure 3.9: Frequency of document accesses versus document ranking (server traces).

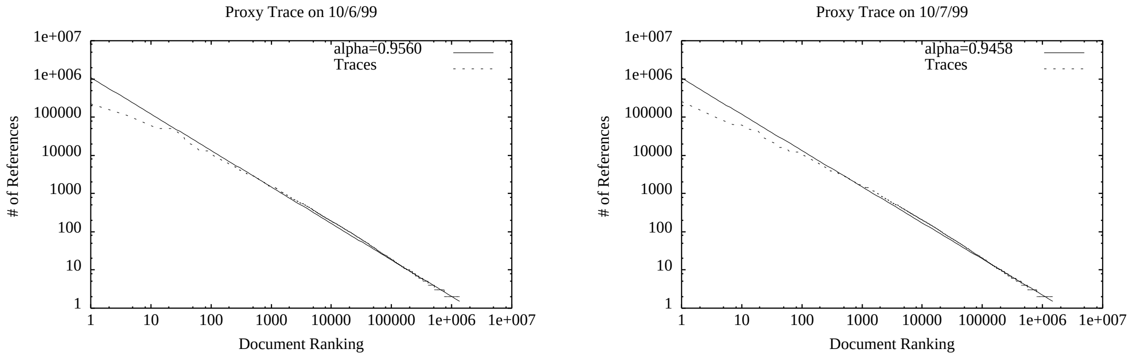


Figure 3.10: Frequency of document accesses versus document ranking (proxy traces).

We make the following observations. The curves for both the server traces and the proxy traces (not shown) fit a straight line reasonably well when ignoring the first 100 documents as in [22]. The few most popular documents' popularity deviates from the straight line especially in the left of Figure 3.9. The remaining documents' popularity fits a straight line reasonably well. The straight line on the log-log scale implies that the request frequency is proportional to $1/i^{\alpha}$. The values of α are obtained using least square fitting, excluding the top 100 documents (as in [22]), and also excluding the flat tail. This shows that the finding of previous proxy-based studies, that Web accesses follow a Zipf-like distribution, extends to accesses seen by a server as well.

However, the range of α values corresponding to our server traces is consistently and significantly higher than that reported in previous proxy-based studies. Specifically, the value of α in the server traces is typically around 1.4 - 1.6, with lowest being 1.397 and highest being 1.816. In contrast, proxy-based studies have reported a much smaller α, typically less than 1.0.

Moreover we find that the value of α is the highest (1.816) on 12/17/98, when there was an unusual event of global interest (Operation Desert Fox). This is as we would expect. During such period users from all over the world are interested in a small set of pages related to the event, making hot documents extremely hot and cold documents colder than usual. So the difference in the hit count of hot pages and cold pages is enlarged, making α larger.

To further study the impact of different α values, we also plot the cumulative distribution of requests to popular documents. Figure 3.11 shows the cumulative probability of access for the top r% of documents for the server traces from 12/17/98 (the highest α due to flash crowds) and 8/5/98 (a "normal" day), and the Microsoft proxy traces from 10/6/99. From the server traces, we observe that on 12/17/98 the top 2% of the documents account for 90% of the accesses, and on 8/5/98 the top 4% of the documents account for 90% of the accesses. The latter is also observed for the other days of the server traces. In comparison, it takes 36% of the documents to account for 90% of the accesses in the proxy traces, which is also shown in [22]; and it takes 10% of the documents to account for 90% of the accesses in the traces from less popular servers collected in 1994 and 1995 [11].

Hence it is evident that Web accesses seen at the server tend to be more concentrated than those seen at the proxies. Moreover the accesses seen at the popular MSNBC server site tend to be more concentrated than those seen at less popular servers. This implies that techniques such as reverse caching and replication could go a long way in alleviating server load, especially for popular servers.

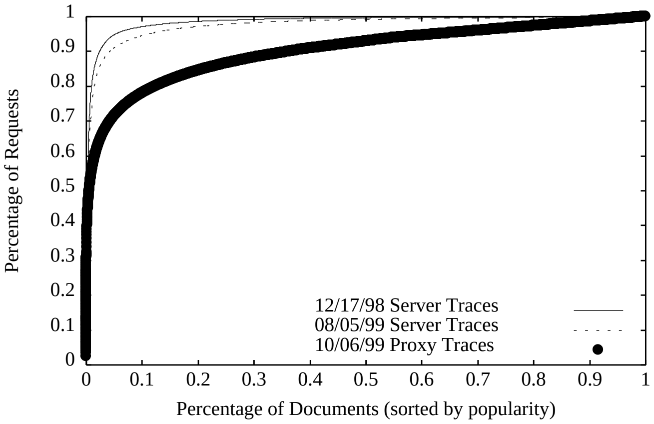


Figure 3.11: Cumulative distribution of requests to documents.

Reason and Implication of Larger α at Web servers

We have observed that the α values in the server traces are consistently and significantly higher than those in the proxy traces. Although our access logs are deficient in that they do not contain accesses to images, such deficiency is unlikely to consistently bias α towards a higher value. As found in [130], the α value for accesses to image and HTML files (based on a large university proxy trace) are 0.80 and 0.76, respectively. This suggests excluding the accesses to images are unlikely to bias α towards a higher value. The following straightforward calculation sheds more light on this.

Suppose, when we exclude image files, two HTML files receive W and W' accesses each. The ranking of the two are R and R' respectively. Then we have

$\alpha = \frac{log(W)-log(W')}{log(R)-log(R')}$. After including image files, we have $\alpha = \frac{log(W)-log(W')}{log(k+i+R)-log(k+R')}$, where $k \geq 0$, and $i \geq 0$. Adding k to the original ranking of both objects is because including more files can move the original files' ranking behind (e.g., once image files are included, a file with a popularity rank of 100 can have its rank pushed back to say 200). Adding i to one of the object is because adding more files can possibly enlarge the difference in ranking of two files. Now we make the following observations:

- If $i = 0$, α would increase after including image files.
- If i is large enough to offset k, α would decrease after including image files.

Based both on analysis and previous trace-based studies, we conclude that the lack of image access information should not create a systematic bias towards higher α values. Rather we believe that the higher α values we observe is because we have studied accesses seen at servers rather than at proxies. A number of previous studies [22, 133] have shown that popular Web pages are spread almost evenly across hot Web servers. This implies that a proxy aggregating requests to a number of popular servers would have a slower decay in popularity than any individual server. This is illustrated by the following simplified calculation.

Suppose a proxy accesses s Web servers. The Web server i has n_i documents, with decay coefficient α_i. In addition, it receives A_i accesses to its most popular Web page. Then according to the Zipf-like distribution, we have

$$log(A_i) = \alpha_i * log(n_i)$$

The decay coefficient at the proxy can be computed as:

$$\alpha_{proxy} = \frac{log(max(A_1, A_2, ..., A_s))}{log(n_1 + n_2 + ... + n_s)}$$

For a simple case when $A_1 = A_2 = ... = A_s = A$, $n_1 = n_2 = ... = n_s = n$, and

$\alpha_1 = \alpha_2 = ... = \alpha_s = \alpha_{server}$, we have

$$\alpha_{proxy} = \frac{log(A)}{log(s*n)} = \alpha_{server} * \frac{log(n)}{log(s*n)} < \alpha_{server}$$

The exact difference between α_{proxy} and α_{server} depends on s (the total number of popular servers). Of course, things are much more complicated in practice: (i) the Web accesses do not strictly follow a Zipf-like distribution, especially for the most popular documents; (ii) Web servers are very heterogeneous. It is very hard to compute α exactly. On the other hand, as long as it is true that the popular Web pages are spread almost evenly across hot Web servers, the α at the proxy is lower than at each individual popular Web server.

In summary, we observe the access patterns at both the server and the proxy exhibit a Zipf-like distribution. The value of α is much higher for the server accesses than for the proxy. In some cases, the top 2% documents account for 90% of the accesses. In contrast, the accesses seen at the proxy traces are more heterogeneous. They take up to 39% to account for 90% accesses. This suggests that the hit ratio for reverse caching or replication is likely to be significantly higher than that observed with traditional proxy caching, and that caching or replicating a small set of files would cut down server load significantly.

3.4 Summary

In this chapter, we have studied the dynamics of a large commercial Web site, which is consistently ranked among the busiest in the Internet. We have analyzed the dynamics of both the content and the client accesses at this site.

3.4.1 Key Results

Our main findings are:

1. The server content tends to be highly dynamic with around 6000 files created and 24000 modified over a one-week period. For the subset of files that are modified, the duration between successive modifications tends to lie between an hour and 24 hours (i.e., a day).

2. Past modification behavior of a file, if averaged over a sufficient number of samples, yield a rough predictor of future modification behavior. The estimation error is within a factor of two 65% of the time and within a factor of five 90% of the time.

3. The *vdelta* between successive versions of files is often extremely small, implying that modifications made to the (HTML) content tend to be minor.

4. The popularity of files tends to be stable over a timescale of days. Of the 100 most popular documents on a given day, 60–100% remain among the top 100 for up to 5 days. However, the set of domains from which the popular files are accessed tends to change significantly from day to day. For example, there is only a 40% overlap in the set of domains when considering the top 100 files. This may be a consequence of our (fine-grained) definition of a domain, and we are currently exploring this further.

5. Organizational (i.e., domain) membership of clients tends to have a significant (positive) impact on the degree of local sharing. This agrees with the results reported in [133]. However, our study differs from [133] in several important ways: (i) [133] studies the accesses made by a university, while

we study the accesses to a large commercial news site from all over the world. The clients in our traces are thus more heterogeneous; (ii) Our domains correspond to (often) large and diverse DNS level-two domains whereas [133] defines “domains” to be the rather small and much less diverse departments within a single university; (iii) We also find when there is a globally-interesting event (such as Operation Desert Fox in December 1998) that cuts across organizational boundaries, the organizational membership is no longer significant in affecting the users’ accesses pattern.

6. File popularity tends to follow a Zipf-like distribution. The parameter α seen at the server is in the range 1.4–1.6, which is much larger than what has been reported in the literature (based on the analysis of proxy logs). The large value of α implies, for instance, that just the top 2% of documents could account for 90% of the accesses.

7. The popularity of most documents tends to drop off with age. However, some documents tend to maintain their popularity for a significant length of time.

8. The majority of first-time accesses from a domain are to unpopular documents at least a day old.

3.4.2 Broad Implications

Our study of both server content and access dynamics has broad implications for the future evolution of the Web.

Implications for Cache Consistency Algorithms

We find that files are modified frequently (Section 3.2.2) and that a large part of a user's requests are to modified files (Section 3.3.1). For instance, over half of the repeated accesses (accesses to files previously requested by the same domain) are to a modified file. This pattern of file modification and access underscores the importance of having efficient cache consistency control mechanisms (e.g., [39] [136]). The modification history of a file can yield a rough predictor (Section 3.2.3) for adaptive TTL-based schemes (e.g., [63]).

Second, the *vdelta* between successive versions of files is often extremely small (Section 3.2.4), so techniques such as delta encoding [95] will be very effective.

Implications for Prefetching and "Push"

Our study has several implications for prefetching and server-based "push".

First, file creation tends to be a frequent event (Section 3.2.1). When a popular news story is updated, it may be assigned a new file name. The absence of past access history makes the task of prefetching or preemptively pushing out newly created content challenging. Clever techniques, such as using the access history of the surrounding pages based on Web linkage structure to make predictions, may be needed.

Second, the high frequency of file modification and accesses to modified files suggests that prefetching or pushing such files would be very useful. Moreover, since the popular objects stay popular for a relatively long period of time (Section 3.3.2), it makes sense to prefetch (or push) previously popular files that have undergone modification.

Lastly, the domain membership of clients is of significance. Clients belonging to the same domain are more likely to share requests than clients picked at random (Section 3.3.3). On the other hand, the stability of interest groups is not very high (Section 3.3.2), so it may be challenging to have server push documents discriminatively to the clients. In future work, we plan to investigate this issue further.

Implications for Web Caching

The implications of our findings for Web caching are twofold.

On the positive side, we find the accesses seen by the Web server are more concentrated to a small set of documents than the previous proxy-based studies, with a higher α (Section 3.3.4). This implies caching (or replicating) a small number of popular documents may potentially reduce server load and/or Web traffic significantly.

On the negative side, it appears that first-time misses, though as high as 40% of the accesses [127], would be hard to cut down significantly. This is because most first-time accesses tend to be to old and unpopular documents (Section 3.3.1), so it is very hard to accurately predict such accesses. Nevertheless, prefetching or pushing modified objects to users is still beneficial due to the high frequency of modification rate and access rate. Therefore, we believe designing efficient cache consistency algorithms, and making more (hitherto uncachable) objects cachable are two major directions to improve the efficiency of Web caching. Replication done under server control may be a viable solution to both issues.

Based on the insights obtained from the workload study, in the next chapter, we design algorithms to provision content distribution networks efficiently.

Chapter 4

Placement of Web Server Replicas

With the explosive growth of the World Wide Web, popular Web sites receive an enormous share of Internet traffic. These sites have a competitive motivation to offer better service to their clients at lower cost. To address the need, there has been an increasing trend toward outsourcing content distribution to commercial hosting services such as Akamai [3], Exodus [51], Digital Island [44], etc. Existing work on CDNs has primarily focused on techniques for efficiently redirecting user requests to appropriate CDN servers to reduce request latency and balance load. However, little attention has been given to the development of placement strategies for Web server replicas to further improve CDN performance. Placement strategies are important because appropriate placement of server replicas benefits content providers by reducing latency for their clients, and benefits ISPs by reducing bandwidth consumption.

In this chapter, we propose several algorithms that can automate the server placement decision. More specifically, we consider the following scenario. A popular Web site aims to improve its performance (e.g., reducing its clients' perceived latency) by pushing its content to some hosting services. The problem is to choose

M replicas (or hosting services) among N potential sites ($N > M$) such that some objective function is optimized under a given traffic pattern. The objective function can be minimizing either its clients' latency, or its total bandwidth consumption, or an overall cost function if each link is associated with a cost.

In our study, we assume that each client uses a single replica (of course, multiple clients can use the same replica). In other words, a client gets all of its content from the same replica. So our analysis of replica placement focuses on the *traffic load* generated by the clients while ignoring what content is actually downloaded by clients. While our assumption is not entirely realistic – for example, a CDN such as Akamai would, in general, have *partial* replicas and direct clients to different replicas depending on what content is accessed – it enables us to project into the future when falling storage costs might make it feasible for each replica to be a *complete* replica. In such a setting, a client may well be directed to a single replica for most or all of its accesses.

We evaluate the performance of the various placement algorithms by simulating their behavior on synthetic and real network topologies and several access traces from large commercial and government Web servers. As far as we know, this is the first experimental study on this subject. We also address a number of practical issues when using these algorithms online in a content distribution network, and study the sensitivity of the placement algorithms to imperfect information about client workload characteristics. Based upon our results, we conclude that a greedy algorithm for solving the Web server replica placement problem can provide content distribution networks with performance that is close to optimal. Although the greedy algorithm depends upon estimates of client distance and load predictions, we find that it is relatively insensitive to errors in these estimates and therefore is a viable algorithm for use in the general Internet environment where workload information will always be imperfect.

The rest of the chapter is organized as follows. We describe graph theoretic formulations to the placement problem in Section 4.1, and present a number of placement algorithms in Section 4.2. Then in Section 4.3 and Section 4.4, we describe our simulation methodology and performance results. In Section 4.5, we discuss a number of practical issues when using the algorithms. We summarize our results in Section 4.6.

4.1 Graph Theoretic Approaches

In this section we review two graph theoretic approaches that can help us determine the number and the placement of Web replicas given the network topology and the users' demands. In the following, we use the terms facilities, centers, and replicas synonymously. We study two variants of the center placement problem: one is the facility location problem, and the other is the minimum K-median problem. Both problems are NP-hard [99]. However, there are constant-factor approximation algorithms for the metric variants of both problems, where the metric variants require that the distance function c is non-negative, symmetric, and satisfies the triangle inequality.

4.1.1 Facility Location Problem

The facility location problem is defined as follows. Given a set of locations i at which facilities may be built, building a facility at location i incurs a cost of f_i. Each client j must be assigned to one facility, incurring a cost of $d_j c_{ij}$ where d_j

denotes the demand of the node j, and c_{ij} denotes the distance between i and j. The objective is to find a solution (i.e., both the number of facilities and the locations of the facilities) of the minimum total cost.

There have been a number of approximation algorithms developed for this NP-hard problem. Throughout the chapter, a *ρ-approximation algorithm* is a polynomial-time algorithm that always finds a feasible solution with an objective function value within a factor of ρ of optimal. Hochbaum [66] showed that the greedy algorithm is an $O(\log n)$-approximation algorithm, and provided instances to verify that this analysis is asymptotically tight. [118] developed the first approximation algorithms with a constant performance guarantee for a number of metric facility location problems. They gave a 3.16-approximation algorithm, which was subsequently improved by Guha & Khuller [61] and by Charikar & Guha [29] who gave 2.41 and 1.728-approximation algorithms, respectively.

4.1.2 Minimum K-Median Problem

The minimum K-median problem is stated as follows. Given n points, we must select K of these to be centers (facilities), and then assign each input point j to the selected center that is closest to it. If location j is assigned to a center i, we incur a cost $d_j c_{ij}$. The goal is to select the K centers so as to minimize the sum of the assignment costs. The main difference between the K-median and facility location problems is that, in K-median, there are no costs for opening centers. Instead, a number K is specified as an input that is an upper bound on the number centers that can be opened.

[30] gave the first constant-factor approximation algorithm, a $6\frac{2}{3}$-approximation algorithm, for solving the minimum K-median problem in the metric space. Jain &

Vazirani [72] and Charikar and Guha [29] subsequently improved this initial result, giving 6- and 4-approximation algorithms, respectively.

4.1.3 Capacitated Versions

The formulations of the facility location problem and minimum K-median problem given above do not constrain the amount of service that can be provided at any center. There are capacitated variants that do constrain the service capacity at centers, requiring that each facility serve no more requests than the capacity defined at that location. The worst-case performance bound for the capacitated variants are considerably worse than for the non-capacitated versions [32, 30].

Depending on different constraints and cost functions to be optimized, replica placement can be formulated as either an uncapacitated/capacitated facility location problem, or an uncapacitated/capacitated minimum K-median problem.

4.1.4 Summary

In the rest of this chapter, we consider the formulation of the uncapacitated minimum K-median problem. That is, we restrict the maximum number of replicas, but do not restrict the number of requests served by each replica. We believe that this is a reasonable formulation because increasing the number of replica sites is significantly more difficult than increasing the capacity of a site. The maximum number of replicas is usually given a priori for cost and administrative reasons, whereas the capacity constraint on the replica can be overcome by adding more machines. [1]

[1] If the capacity of replicas needs to be taken into account, we can use the capacitated minimum K-median problem formulation, and the algorithms described below will still apply.

We also ignore the cost of placing replicas for the following reasons. If our objective function is to minimize network bandwidth consumption, we can ignore the replication traffic (i.e., the traffic associated with managing the replicas, distributing content to the replicas, etc.) since it is typically orders of magnitude smaller than the traffic generated by users' requests. Furthermore, since most content distribution networks (e.g., Akamai) that replicate content have their own private high-speed networks, the bandwidth consumption incurred during the replication is usually not a major concern. On the other hand, if our objective function is to optimize another performance metric, such as users' perceived latency, then it is unclear how to incorporate the replication cost (in the unit of network bandwidth) into the objective function in a different unit (such as time in the case of client latency).

As in [85], we fix the origin server to be one of the replica sites. However, including or excluding the original server is not a fundamental choice and has little impact on our results.

4.2 Placement Algorithms

In this section, we present a number of algorithms for solving the minimum K-median problem. The objective is to minimize the total cost of all the requests. We define the cost of a request from node i to node j as the distance between the two nodes, where the distance can reflect any performance metric we want to optimize, such as latency, hop counts, or the economic cost of the path between two nodes (assuming there is a cost associated with the links on the path). The algorithms work the same regardless of what metric is used.

4.2.1 Tree-based Algorithm

Li et al. proposed a placement algorithm in [85] based on the assumption that the underlying topologies are trees, and modeled it as a dynamic programming problem. The algorithm was originally designed for Web proxy cache placement, and it is also applicable for Web replica placement. At a very high level, they divide a tree T into several small trees T_i, and show that the best way of placing $t > 1$ proxies in the tree T is to place t'_i proxies the best way in each small tree T_i, where $\sum_i t'_i = t$. The algorithm is shown to find an optimal placement when the underlying topologies are trees, and clients request from the proxy on the path toward the Web server, that is, clients cannot request from a sibling proxy. However, these two assumptions can prune possibly better placement choices. As shown in Section 4.4, the optimal solutions under these assumptions are usually not as good as the solutions found by the greedy and hot spot heuristics (without the assumptions), which are described later in this section.

4.2.2 Greedy Algorithm

The basic idea of the greedy algorithm is as follows. Suppose we need to choose M replicas among N potential sites. We choose one replica at a time. In the first iteration, we evaluate each of the N potential sites individually to determine its suitability for hosting a replica. We compute the cost associated with each site under the assumption that accesses from all clients converge at that site, and pick the site that yields the lowest cost. In the second iteration, we search for a second replica site which, in conjunction with the site already picked, yields the lowest cost. In general, in computing the cost, we assume that clients direct their accesses to the nearest replica (i.e., one that can be reached with the lowest cost).

We iterate until we have chosen M replicas. Figure 4.1 shows the pseudo-code.

```
Greedy(N, M)
{
    for i := 1 to M {
        foreach remaining replicas R
            cost(R) := cost after placing an additional replica at R;
        select the replica with the lowest cost
    }
}
```

Figure 4.1: Greedy Algorithm.

4.2.3 Random

The random algorithm is oblivious to client workload, and randomly chooses M replicas among N potential sites from a uniform distribution. To improve performance, we execute the algorithm several times – in our simulations, we execute over 10 times, and pick the random assignment that yields the lowest cost.

4.2.4 Hot Spot

The hot spot algorithm attempts to place replicas near the clients generating the greatest load. It sorts the N potential sites according to the amount of traffic generated within their vicinity. It places the replicas at the top M sites that generate the largest amount of traffic. We define A's vicinity as the circle centered at A with some radius. In our simulations, we vary the radius from 0 to the maximum distance between any pair of nodes in the graph, and report the best performance over all the radii tested.

4.2.5 Super-Optimal Algorithm

As mentioned earlier, the minimum K-median problem is NP-hard. Computing the exact optimal solution is therefore too computationally intensive to be useful in practice. To evaluate how well the algorithms described above perform, we compute a (fairly tight) lower bound on the cost of any feasible solution of our minimization problem. We do so using a *super-optimal* algorithm based on Lagrangian relaxation with subgradient optimization [40]. As its name suggests, the solution produced by this algorithm may be better than optimal because it may not be feasible. Nevertheless, it serves as a useful data point for comparison. More specifically, the K-median problem can be stated as the following integer program, where the 0-1 variable y_i, $i \in N$, indicates whether the location i is selected as a center, and the 0-1 variable x_{ij}, $i, j \in N$, indicates whether location j is assigned to the center at i:

minimize $\sum_{i,j \in N} d_j c_{ij} x_{ij}$ (1)

subject to $\sum_{i \in N} x_{ij} = 1$ for each $j \in N$, (2)

$x_{ij} \leq y_i$ for each $i, j \in N$, (3)

$\sum_{i \in N} y_i \leq k$, (4)

$x_{ij} \in \{0, 1\}$, for each $i, j \in N$, (5)

$y_i \in \{0, 1\}$, for each $i \in N$. (6)

The set of constraints (2) ensures that each location $j \in N$ is assigned to some center $i \in N$, the set of constraints (3) ensures that, whenever a location j is assigned to a center i, then a center must have been opened at i, and (4) ensures that at most k centers are open.

We then apply Lagrangian relaxation to the integer programming problem, where the constraints (2) are weighted by multipliers and placed in the objective function as follows:

$$L(x, y, u) = \sum_{i,j \in N} d_j c_{ij} x_{ij} - \sum_{j \in N} u_i (\sum_{i \in N} x_{ij} - 1)$$

Then we use the subgradient method given in Held *et al.* [132] to compute the super-optimal bound. It is an iterative procedure that begins with a specified multiplier value u^0 and generates a sequence of multiplier values u^k. On iteration k the Lagrangian problem is solved with multipliers u^{k-1} and a new value u^k is determined from u^{k-1} and the Lagrangian solution using the rule in [132]. At each iteration an upper bound on the optimal value is available. In order to obtain the lower bound in our minimization problem, we use the subgradient method on $-L$. In our simulations, we use 1000 iterations for 100 nodes topologies, and 200 iterations for all other topologies. To get a tighter bound, for a specific instance of problem we use three random values to initialize the multiplier u^0, and obtain a super-optimal bound from each value of u^0. The maximum of the three is used as the lower bound.

4.2.6 Summary

Table 4.1 lists the computational time of various algorithms for selecting M replicas among N potential sites. If only a handful of potential hosting sites is available, the cost of the computationally complex algorithms may not be significant. However, in our analysis, we consider clusters defined by address prefixes, which will be explained in Section 4.3.2, as potential replica sites. In this case, N is on the

Table 4.1: Comparison of computational time of various algorithms.

Tree-based [85]	$O(N^3M^2)$
Greedy	$O(N^2M)$
Random	$O(NM)$
Hot Spot	$N^2 + min(NlognN + NM)$

order of 100,000 (the number of address prefixes in the Internet), so clearly the computational complexity of the replica placement algorithm becomes very significant. To reduce the computational cost, we consider only the top, in terms of requests generated, few hundreds or few thousands of clusters. Since these top clusters contribute most traffic, as shown in Section 4.3.2, ignoring the requests from unpopular clusters has little effect on the results.

In the following sections, we will compare the above algorithms with the super-optimal algorithm using the real Web traces and network topologies.

4.3 Simulation Methodology

To evaluate the performance of the various algorithms presented in this chapter, we simulate the behavior of the algorithms on a variety of network topologies and Web workloads. In this section, we discuss the network topologies and Web workloads that we use in our evaluations. We then describe the performance metric that we use as a basis for comparing the algorithms.

4.3.1 Network Topology

In our simulations, we use both randomly generated network topologies and the real Internet topologies derived from BGP routing tables.

We generate two types of random network topologies: random trees and random graphs. The primary goal for studying performance on the tree structure is to determine how the optimal tree-based algorithm compares to the other algorithms. To generate random trees, we wrote a simple program that takes 3 parameters: the total number of nodes, the maximum distance between any two nodes, and the maximum degree of a tree node. Starting from the root node, we recursively create random children until the total number of nodes specified is reached. In our simulations, we use 100-node and 300-node trees, and we set the maximum distance to 10 and the maximum node degree to 10, 15, and 20. For each parameter setting, we generate three different trees.

To generate random graphs, we use the GT-ITM internetwork topology generator [24]. In particular, we use three network models: pure random, Waxman, and Transit-Stub. In the pure random model, vertices are distributed at random locations in a plane, and an edge is added between a pair of vertices with probability p. In the Waxman model, the probability of an edge from u to v is given by

$$P(u, v) = \alpha e^{-d/(\beta L)},$$

where $0 < \alpha$ and $\beta \leq 1$ are parameters of the model, d is the Euclidean distance from u to v, and L is the maximum distance between any two nodes. The Transit-Stub model generates hierarchical graphs by composing interconnected transit and stub domains; see [137] for further details.

We use a wide range of parameters for each network model. For each parameter setting, we generate three different topologies. We do not claim that these network models and parameters we use are representative for the Internet topology. Instead, our goal is to make the generated topologies as rich as possible by using multiple models with a wide range of parameters. As we will show in Section 4.4, the performance of the placement algorithms is similar across different network models and parameters.

We also construct a simplified model of the actual Internet topology using BGP routing data from a set of seven geographically-dispersed BGP peers. Each BGP routing table entry specifies an AS path, $AS_1, AS_2, \cdots, AS_n$, to a destination address prefix block (AS_1 corresponds to the BGP peer and AS_n corresponds to the destination address prefix block). We construct an AS-level topology graph of the network using the AS paths. The AS path $AS_1, AS_2, \cdots, AS_n$ yields edges between adjacent nodes (ASs) in the path (e.g., $(AS_1, AS_2), (AS_2, AS_1), (AS_2, AS_3), etc.$). We map individual clients and address prefix blocks to their corresponding AS nodes in the topology graph, and assign the distance between two nodes as the AS hop counts between the two nodes.

While not very detailed, an AS-level topology at least partially reflects the true topology of the Internet. Furthermore, recent study [89] has shown the AS hop count of a path is a decent indicator of the path's proximity, reliability, and stability.

4.3.2 Web Workload

To evaluate the algorithms on realistic traffic patterns, we use the access logs collected at the MSNBC server site [96], during three periods, as shown in Table 4.2. For diversity, we also use the traces collected at ClarkNet [35] and NASA Kennedy Space Center in Florida [97] during 1995. Table 4.2 shows the detailed trace information. We use the workload in one day or 3 hours (for the August 1999 traces) to parameterize one simulation setup.

We use the access logs in the following way. First, we use the approach proposed by Krishnamurthy *et al.* in [79] to cluster the Web clients that are topologically

Table 4.2: Access logs used.

Trace ID	Web Site	Period	Duration
1	MSNBC	8/3/99 - 8/5/99	9 am - noon
2	MSNBC	9/27/99 - 10/1/99	All day
3	MSNBC	10/7/99 - 10/14/99	All day
4	ClarkNet	9/4/95 - 9/10/95	All day
5	NASA	7/1/95 - 7/31/95	All day

close together. Their method is based on the information available from BGP routing table snapshots, and they show that it can automatically identify clusters for 99.9% of the clients in a wide variety of Web server logs. The identified clusters meet the proposed validation tests in over 90% of the cases, significantly outperforming a heuristic that assumes a fixed-length, 24-bit network prefix.

To use their method for clustering clients, we obtained the complete BGP routing tables from seven geographically and topologically diverse ISPs [69]. For each client IP address in the access logs, we find its best matching prefix in the union of the routing tables. All the clients whose IP addresses have the same best prefix match belong to the same cluster. Figure 4.2 plots the number of requests generated by each cluster. As we can see, in the 8/3/99 MSNBC trace, the top 10, 100, 1000, and 3000 clusters account for about 23.55%, 44.86%, 77.96%, and 93.97% requests, respectively. The other server traces have similar results, though the NASA traces are a little more concentrated: the top 10, 100, 1000, and 3000 clusters account for about 29.67%, 51.96%, 85.39%, and 97.37% requests, respectively.

For a network topology of a specific size, say 100 nodes, we choose the top 100 clusters in the traces and map them randomly to the nodes in the graph. Assigning

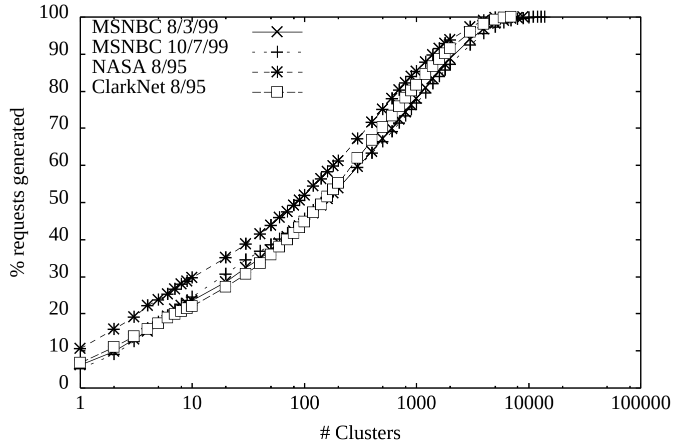


Figure 4.2: The CDF of the number of requests generated by the Web clusters defined by address prefixes.

a cluster C_i to a node P_i in the graph means that the weight of the node P_i is equal to the number of requests generated by the cluster C_i. For each network topology and access log, we make three different random assignments from the clusters to the nodes in the graphs.

4.3.3 Performance Metric

To compare the performance of the algorithms on the various network topologies and access logs, we use the ***relative performance*** of the algorithms as a metric. We define the relative performance as the ratio between the cost of the feasible solution found by the algorithm to the cost determined by the super-optimal algorithm. The relative performance is an appropriate metric, since it reflects the cost we want to minimize. The normalization step is to show how far away we are from the super-optimal, and does not alter the metric to be minimized. The smaller the value of the relative performance, the better the algorithm performs. A relative performance of 1 implies the algorithm finds an optimal solution, but the optimal solution need not necessarily have a relative performance of 1, since the super-optimal solution may not be achievable.

4.4 Simulation Results

In this section, we evaluate the performance of the various placement algorithms on a variety of synthetic and realistic network topologies using the Web server traces.

4.4.1 Random tree topologies

First, we evaluate the performance of the placement algorithms on the tree topologies. More specifically, we run each placement algorithm in hundreds of simulation runs and examine the performance of the algorithm across all simulation runs. Each simulation run is parameterized by (i) the Web server trace, (ii) the network topology, (iii) the mapping from the cluster to the node in the simulation topology, and (iv) the number of replicas to choose. We evaluate the algorithms on 100-node and 300-node trees using Web traces 1 and 3 listed in Table 4.2. We use three different random assignments from the clusters defined by address prefix to the nodes in the trees. We then vary the number of replicas to place from 1 to 80 for the 100-node trees, and from 1 to 100 for the 300-node trees.

Figure 4.3 shows the cumulative distribution (CDF) of the relative performance of the algorithms on tree topologies. A point on a curve denotes the percentage of all simulation runs that result in a particular relative performance or better. For example, in Figure 4.3a, the first square on the curve for the greedy algorithm shows that 9% of the simulation runs of the greedy algorithm on a 100-node tree topology had a relative performance of 1. As we can see, the greedy algorithm and the tree-based algorithm perform the best, with the greedy algorithm slightly better. The hot spot algorithm has a performance in between these two and the random algorithm, which clearly has the worst performance. We quantify the differences in relative performance of the algorithms in the next set of graphs.

Figure 4.4 shows the minimum (best case), maximum (worst case), and median values of the relative performance of these algorithms over all simulation runs, where the tree-based, greedy, random, and hot spot algorithms are numbered algorithm 1, 2, 3, and 4, respectively. On average, both the greedy and tree-based algorithms are within 5% worse than the super-optimal algorithm for 100-node trees, and within 20%–30% worse than the super-optimal algorithm for 300-node trees. The hot spot algorithm has a relative performance that is about 30% worse than the super-optimal algorithm. The random algorithm performs considerably worse than the others. This is also evident from its CDF curve shown in Figure 4.3, which has a very gradual slope. For the three graph sizes, 50% of the simulation runs for the random algorithm have a relative performance of at least 2.5. Note also that the relative ranking of these algorithms is consistent across all the tree topologies and Web traces tested.

The reason that even in tree topologies the tree-based algorithm is not the best performer is that it assumes clients can only direct requests to replicas on the path toward the Web server. This assumption eliminates some potentially better placement choices.

4.4.2 Random graph topologies

We also evaluate the performance of the placement algorithms on random graphs generated by the GT-ITM topology generator. As with the tree topologies, we

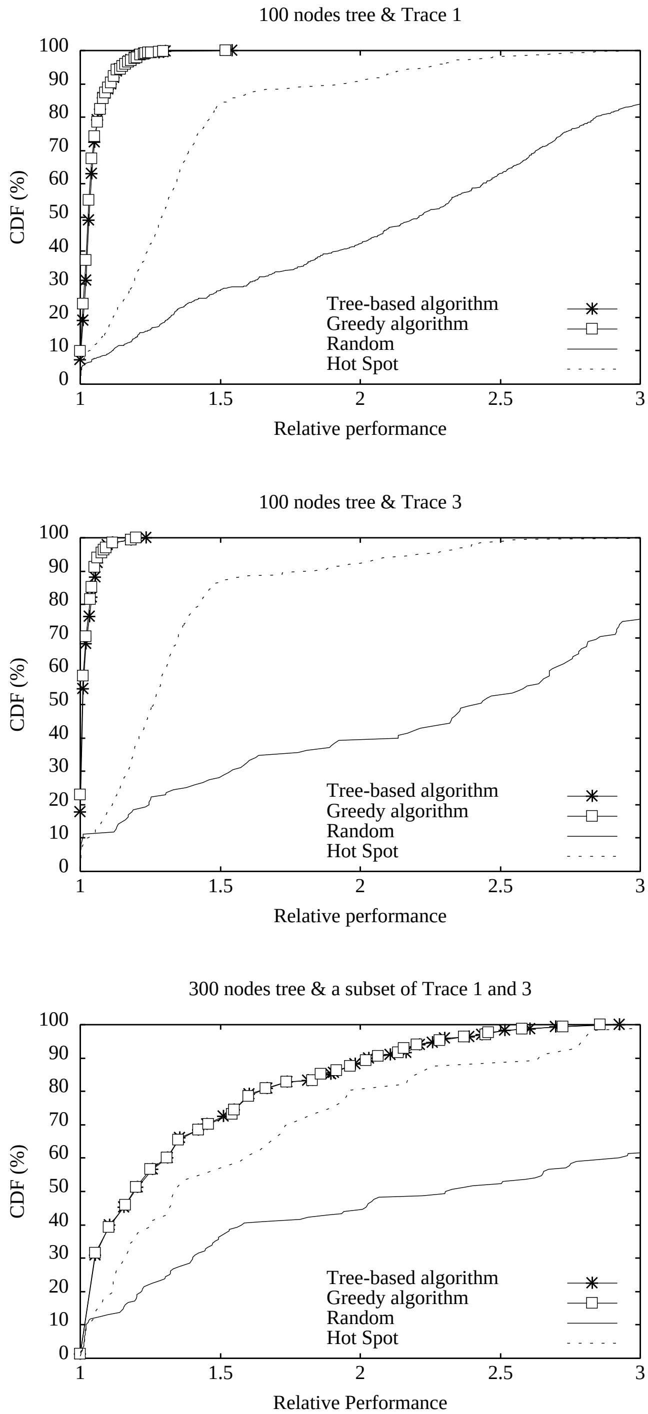


Figure 4.3: The CDF of relative performance across all simulation runs of the placement algorithms on tree topologies.

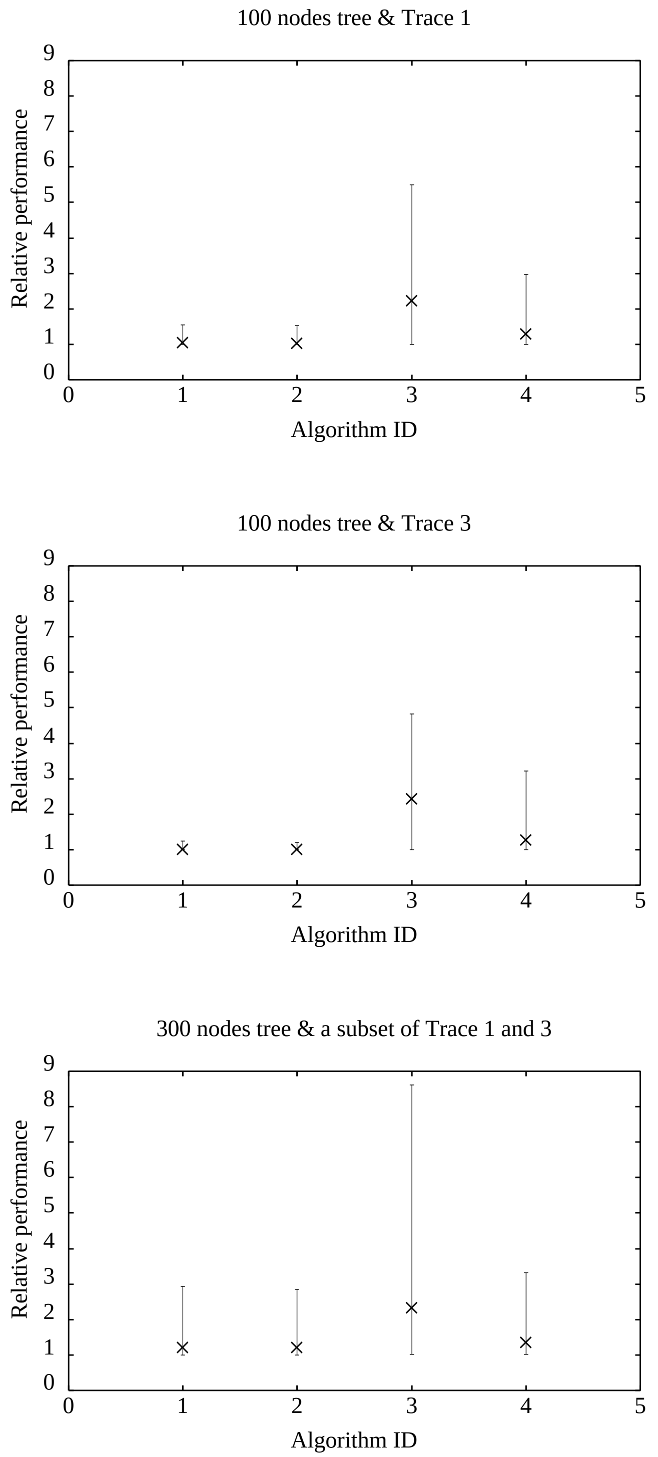


Figure 4.4: The performance of the placement algorithms on tree topologies using errorbars. The lower and upper bounds, and the X mark on each errorbar correspond to the minimum, maximum, and median, respectively, of the relative performance of the corresponding algorithm. The tree-based, greedy, random, and hot spot algorithms are respectively numbered 1, 2, 3, and 4 in the graph.

run each algorithm in hundreds of simulation runs and examine the performance of the algorithms across all simulation runs. We vary the number of replicas to place from 1 to 80 for the 100-node graphs, from 1 to 100 for the 300-node graphs, and from 1 to 200 for the 1000 and 3000-node graphs. For every graph size, we use three network models with different parameters, as described in Section 4.3.1. We plot the CDF of the relative performance of the different placement algorithms across all simulation runs in Figure 4.5. We show the minimum, maximum, and median of the relative performance across all simulation runs using errorbars in Figure 4.6, where the algorithms are numbered as in Figure 4.4.

Before describing the results in the graphs, we make the following observations. First, the tree-based algorithm requires the underlying topology to be a tree. For our evaluation of the tree-based algorithm on general graphs, we generate three random spanning trees for a given graph, where all the spanning trees are rooted at the original server node. Then we run the algorithm on each of the trees. The three adjacent errorbars for $Algorithm\ ID = 1$ in Figure 4.6 correspond to the performance obtained using the three different spanning trees. Second, we only report the performance of the tree-based algorithm for 100-node and 300-node topologies since it takes too long to run on topologies with 1000 or more nodes. For example, it takes over 11 hours to place 5 replicas among 1000 potential sites on an UltraSparc machine with a 500 MHz CPU and 4 GB of memory. As a result, we conclude that the tree-based algorithm is not practical for making real-time placement decisions when the network size grows to thousands of nodes. In comparison, for the same scenario, the greedy, hot spot, and random algorithms take less than 1 minute to run.

Compared to the super-optimal algorithm, the greedy algorithm performs within a factor of 1.5 in the median cases, and around a factor of 4 in the maximum cases. These results are significantly better than all of the other algorithms, including the tree-based algorithm. Another interesting observation is that the hot spot algorithm is often better than the tree-based algorithm on the general graphs. The random algorithm, as before, performs the worst: its median performance is around 2.5 and its maximum relative performance is as high as 11–13.

4.4.3 Internet topology

We also evaluate the performance of the placement algorithms using a model of Internet topology derived from BGP routing tables. In this case, we use AS hop counts as the distance metric between two connected nodes. As shown in Figure 4.7 and Figure 4.8, the ranking of the various algorithms stays the same as in the randomly generated graphs. From the best to the worst in order are the greedy, hot spot, tree-based, and random algorithms. However, the performance difference between the algorithms is smaller than that in the randomly generated graphs. This is because the number of AS hops between any two nodes is not as widely distributed as the distance in the generated topologies. The number of AS hops varies from 0 to 6 for the 100 top ASs (in terms of the number of requests generated to the MSNBC Web server during the periods under study), and from 0 to 9 for the 1000 top ASs. In contrast, the distance between any two nodes in the generated topologies can be different by orders of magnitude.

4.4.4 Effects of imperfect knowledge about input data

The above simulation results are based on the assumption that we have perfect knowledge of the underlying topologies and the number of requests generated from

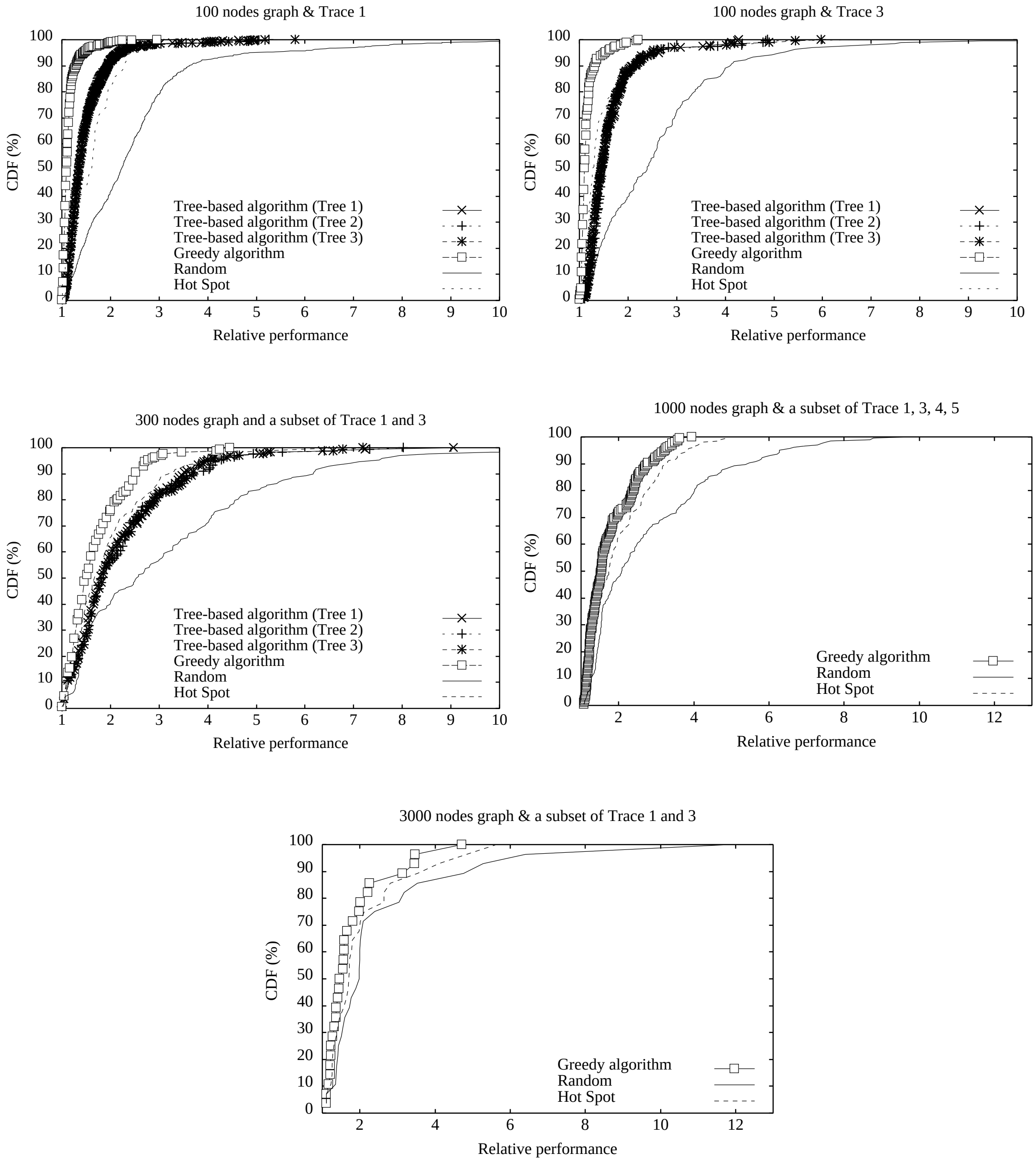


Figure 4.5: The CDF of the relative performance across all simulation runs of the placement algorithms on general graphs.

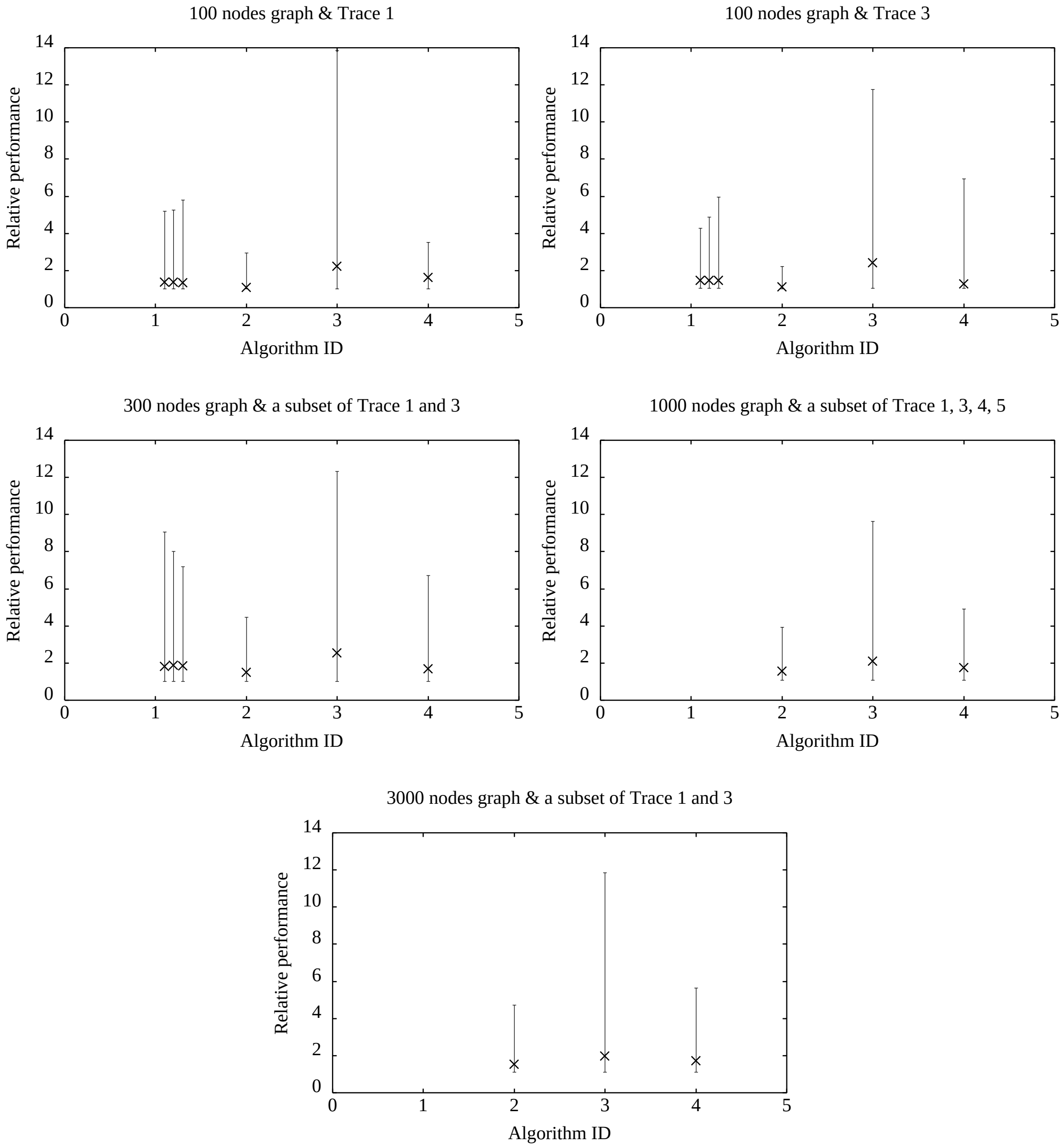


Figure 4.6: A summary of the performance of the placement algorithms on graph topologies using errorbars. The lower and upper bounds, and the X mark on each errorbar correspond to the minimum, maximum, and median, respectively, of the relative performance of the corresponding algorithm. The tree-based, greedy, random, and hot spot algorithms are respectively numbered 1, 2, 3, and 4 in the graph.

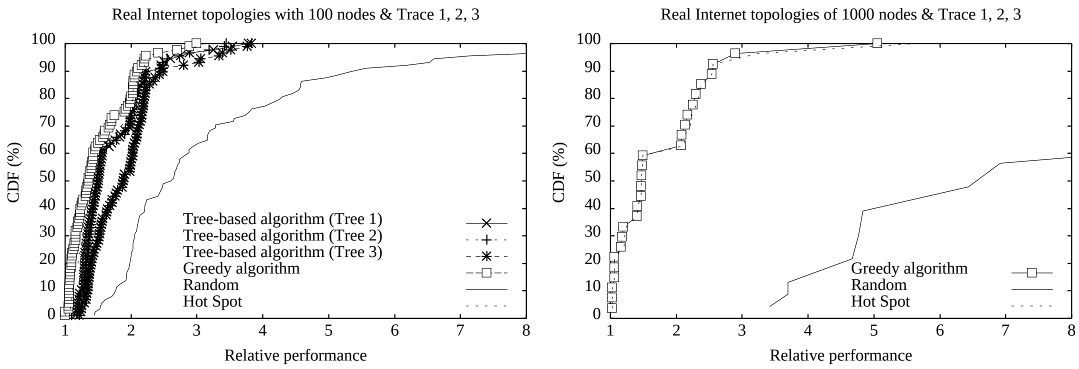


Figure 4.7: The CDF of relative performance across all simulation runs of the placement algorithms on the model of the real Internet topology derived from the BGP routing tables.

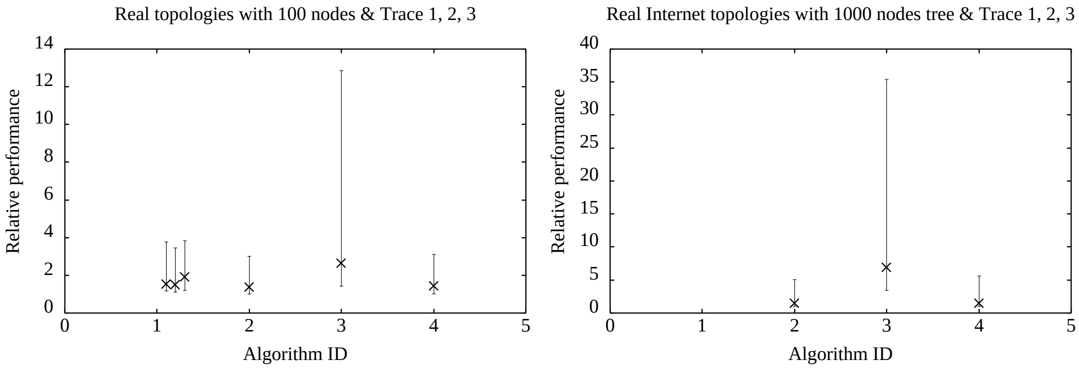


Figure 4.8: A summary of the relative performance of the placement algorithms on the model of the real Internet topology using errorbars. The lower and upper bounds, and the X mark on each errorbar correspond to the minimum, maximum, and median, respectively, of the relative performance of the corresponding algorithm. The tree-based, greedy, random, and hot spot algorithms are respectively numbered 1, 2, 3, and 4 in the graph.

each node. In practice, we do not have perfect knowledge about these inputs, but only have rough estimates. In this section, we examine how imperfect knowledge about the input data affects the placement decision. In particular, we want to find out if the placement decision based on inaccurate information will still be useful, and how far its performance deviates from that obtained using perfect knowledge.

Our approach is to salt the input data with random noise of uniform distribution, and vary the amount of noise added to the input data. This is done in two ways: (1) we perturb the volume of requests from a client by up to a factor of 2 (i.e., if the true number of requests is d, the perturbed value ranges between $\frac{d}{2}$ and $2d$), and (2) we perturb the distance, c_{ij}, between two nodes i and j by up to a factor of 4 (i.e., the corrupted distance ranges between $\frac{c_{ij}}{4}$ and $4c_{ij}$). We feed the salted inputs to the placement algorithms, and compute the cost after applying the placement decision to the actual input data. As before, we use relative performance as the metric, defined as the ratio between the cost of the feasible solution found by the algorithms using the salted inputs to the cost determined by the super-optimal algorithm using the actual inputs.

Figure 4.9 shows the minimum, maximum, and median of the relative performance over all the values of the error rates in the distance and load. As we can see, the performance deviation is small. In particular, even with the salted error as high as a factor of 4, the cost of the greedy algorithm is in most cases within a factor of 2 of the super-optimal algorithm when using perfect knowledge. This is also evident from Figure 4.10, which plots the relative performance of the greedy algorithm versus the errors in the input. As we can see, as the error increases, the performance degrades only slightly.

The above perturbation technique on the topology is most useful when our

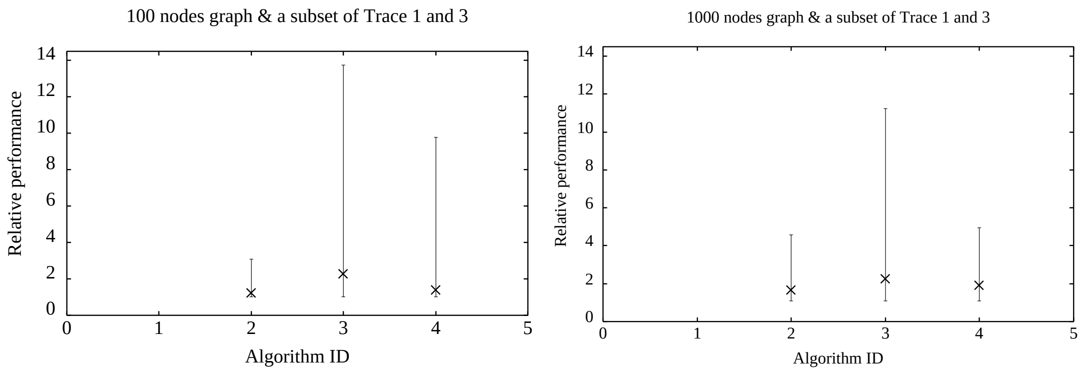


Figure 4.9: The relative performance of the placement algorithms on the graph topologies using errorbars, with both the load and distance information salted with random noise.

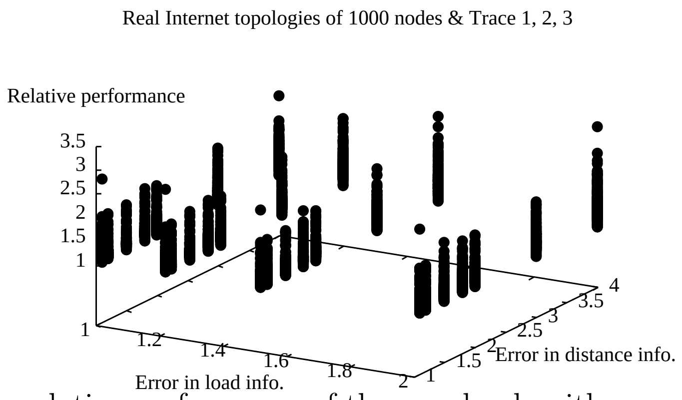


Figure 4.10: The relative performance of the greedy algorithm on the input salted with random noise.

performance metric is the propagation delay or the economic cost of the paths. If our performance metric is AS hop count, we can infer the distance between two nodes by using BGP routing tables as illustrated in Section 4.3.1. However, when the number of BGP peers providing routing information is very limited, we may not have a very accurate AS-level topology map (for example, we do not see all the links).

To study the effect of overlooking some network links on the placement algorithms, we randomly remove 0–50% of the edges in the AS-level Internet topology derived from the BGP routing tables, and use the perturbed topology information in the placement algorithm. Figure 4.11 shows the performance results for the greedy algorithm normalized by the performance of the super-optimal algorithm using perfect topology information for the 10/8/99 MSNBC server trace. As we can see, the performance of the greedy algorithm hardly changes as more edges are removed. In particular, even when the edge removal probability is as high as 50%, the relative performance of the greedy algorithm stays within 2.6. The insensitivity of the greedy algorithm to the edge removal partly comes from the fact that the only topology information that the greedy algorithm (and all other algorithms except the tree-based algorithm) requires is the distance matrix. When testing the distance matrix in more detail, we find that the distance matrix is not sensitive to edge removal. In particular, removing up to 5% of the edges in the graph does not change the distance matrix in our experiments.

4.4.5 Stability of input data

The above section studies the effect of imperfect knowledge on the placement decision. One of the major reasons that we do not have perfect knowledge about

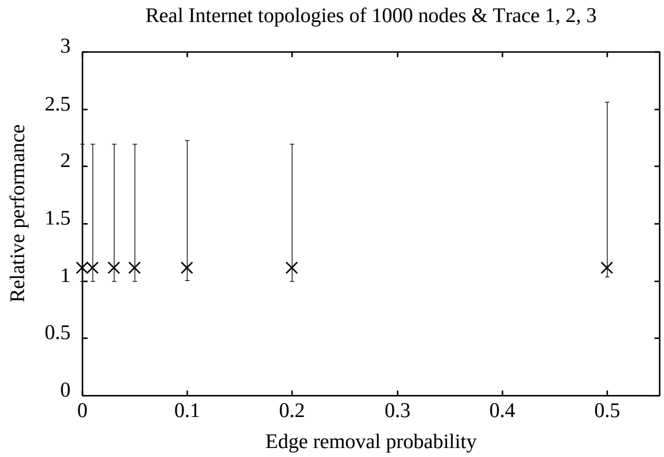


Figure 4.11: A summary of the relative performance of the greedy algorithm during edge removal using errorbars. The lower and upper bounds, and the X mark on each errorbar correspond to the minimum, maximum, and median, respectively, of the relative performance of the greedy algorithm.

the input data is that the input data is changing over time. When making the placement decision for the next 24 hours, ideally we would like to give the placement algorithms the load and network information for the next 24 hours. However, in practice, we can only use the past information to predict the future load and network information. How good such a prediction is can significantly affect the performance of the placement algorithm. In this section, we investigate this issue in detail.

Our evaluation is done in two parts. In the first part, we assume the topology information is accurate but the load information is based on the prediction. In particular, we consider the scenario where we want to make a placement decision for 10/1/99 by using the workload for the previous few days. We predict the load generated from a cluster by averaging its load during the previous n days, where n varies from 1 to 4.

To perform this evaluation, we use Trace 2 listed in Table 4.2, which contains the access logs from 5 consecutive working days (from Monday to Friday). We pick the top 1000 clusters from 10/1/99. (The top 1000 clusters on 9/27/99 – 9/30/99 have more than 90% overlap with those on 10/1/99.) As before, we randomly assign the clusters to the nodes in the randomly generated topologies of various network models and parameters. For each topology and cluster assignment, we simulate the placement algorithms on the actual workload on 10/1/99 and five predicted workloads: (i) the workload of 9/30/99, (ii) the averages of 9/29/99 and 9/30/99, (iii) the averages of 9/28/99 – 9/30/99, and (iv) the averages of 9/27/99 – 9/30/99.

Figure 4.12 shows the CDF of the greedy algorithm's performance across all simulation runs. Here we normalize the performance of the greedy algorithm using the predicted load by its performance using the actual workload on 10/1/99. The lower the normalized performance, the better the prediction is. A normalized performance of 1 means the performance is exactly the same as obtained using the actual workload. As we can see, the performance using the predicted workload comes very close to the performance using the actual workload, within 5% over all cases. Note that, in some cases, the performance using the predictions is slightly better than using the actual workload. This is because the greedy algorithm does not give the optimal performance even when the input data is completely accurate.

For the second part of the evaluation, we use the same strategy as above, but salt the topology information with random noise as described in Section 4.4.4. Figure 4.13 shows the performance results when we perturb the distance between any two nodes by up to a factor of 1.2 and 2. As we can see, the performance deviation from using the accurate load and network information is small: when the perturbation in distance is up to a factor of 1.2 and 2, the deviation is only within 5% and 17%, respectively. Moreover, the performance results are similar

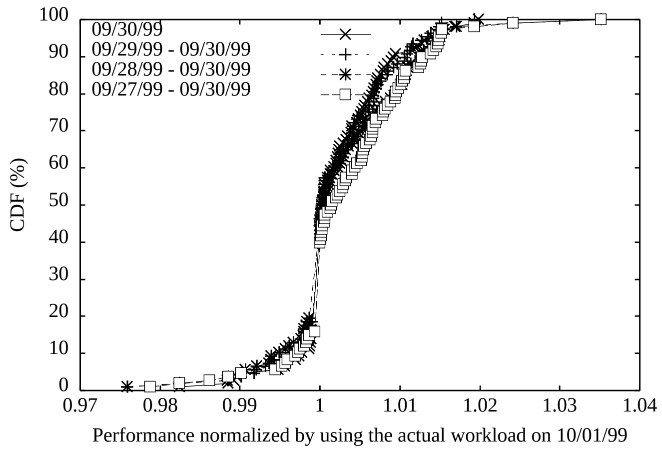


Figure 4.12: The CDF of the greedy algorithm's performance using the predicted workload normalized by its performance using the actual workload across all simulation runs.

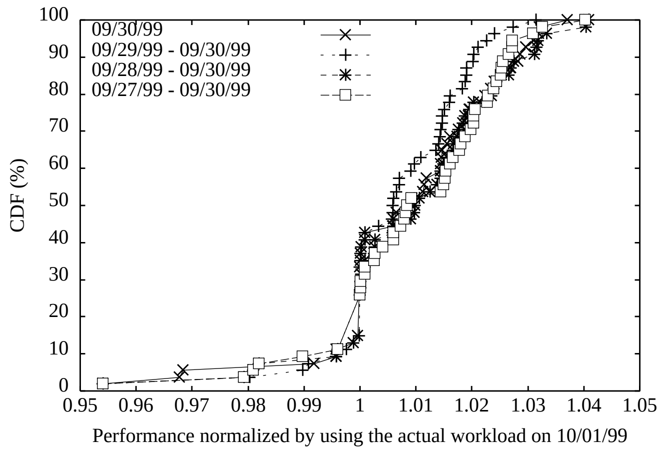
across all the prediction windows tested.

Finally, we have observed significant variation between weekday workloads and weekend workloads, even though they are consecutive in time. This is not surprising, and suggests that we should use past weekday workload information to predict future weekday workloads, and likewise use past weekend workload data to predict future weekend workloads.

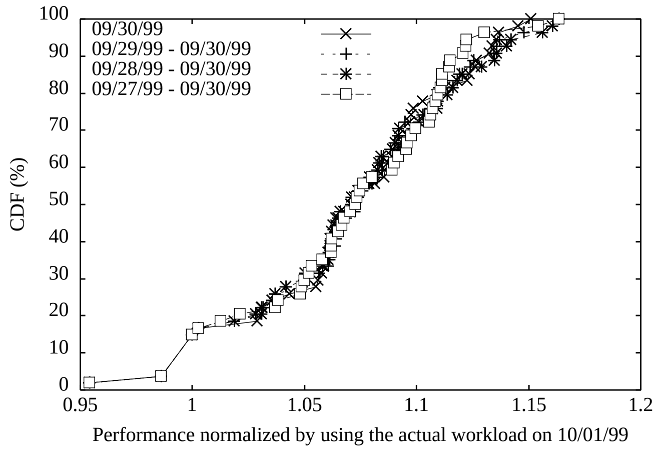
4.5 Practical Considerations

In this section, we discuss ways to obtain the input data for the placement algorithms. As mentioned earlier, the input to the placement algorithms is a graph with weighted nodes and edges. A node's weight represents the amount of traffic initiated by the node, and an edge's weight represents latency, or link cost, or hop count, etc. In order to apply the placement algorithms in practice, we need to be able to obtain both traffic pattern and network topology information in real-time.

Obtaining node weights is relatively straightforward. During re-provisioning,



(a) Perturbation factor is 1.2



(b) Perturbation factor is 2

Figure 4.13: The CDF of the greedy algorithm's performance using the predicted workload normalized by its performance using the actual workload when we also perturb the distance between any two nodes.

the Web server communicates with all the active replicas (i.e., the replicas that serve the requests, as opposed to potential replica sites) about the number of requests generated by all the popular clusters, where clusters are identified using the approach proposed in [79].

The method for obtaining edge weights depends on the performance metric that we want to optimize. Since replication placement is a relatively long-term provision, we believe it is desirable to use the performance metrics that are stable on the order of hours, such as propagation delay, hop count, or economic cost of the path between two nodes.

To approximate the distance between each pair of nodes, we can use BGP routing tables to infer the hop counts between each pair of nodes as described in Section 4.3. An interesting question is how many BGP peers we would need routing information from in order to construct a fairly accurate AS-level topology map. The answer clearly depends on the richness of the connectivity, i.e., the (average) degree of nodes in the topology graph. The greater the degree, the greater the number of BGP peers from which we will need routing information. The worst case is a completely connected graph (which, however, is far from the reality). However, as we show in Section 4.4, the performance of the greedy algorithm is not sensitive to overlooking some network links – its relative performance stays within 2.6 of the super-optimal algorithm even when the edge removal probability is as high as 50%.

A separate question is whether knowing the topology is sufficient for solving the placement problem. In general, we would need some notion of Internet "weather", that is, the network performance between two points, say a client location and a potential replica site. There are several research efforts (e.g., IDMaps [73]) focusing on the problem of constructing such an Internet weather map. If desired, we could, in a straightforward manner, substitute cost metrics derived from an Internet weather map in place of those derived from topology information in our algorithms. Before such a service is widely available, we can also have the Web sites periodically *ping* or *traceroute* a representative client in each identified popular cluster. Since the number of popular clusters is not large, usually around 1 – 3 thousand as is the case with MSNBC Web site, such probing is affordable especially when the provisioning timescale is on the order of hours or longer.

4.6 Summary

In this chapter, we studied the online problem of placing Web server replicas in content delivery networks (CDNs) to minimize the cost for clients to access data replicated on the servers. The cost function is generic, and can represent latency, hop count, the economic cost between the two nodes, etc. We assume that the CDNs have a mechanism for routing requests to replicas [74], and focus entirely on the replica placement problem.

We approach the placement problem by formulating it as a minimum K-median graph theoretic problem. We also give various algorithms for solving the minimum K-median problem, including a super-optimal algorithm for providing optimality bounds. We then evaluate the performance of the algorithms by simulating their behavior on synthetic and real network topologies and several access traces from large commercial and government Web servers. As far as we know, this is the first experimental study on this subject.

We also address a number of practical issues when using these algorithms online in a content distribution network. In particular, we study the sensitivity of the algorithms to imperfect knowledge about the input data (such as measurement error for client latency), the stability of the inputs over time (for predicting future behavior based upon past observed behavior), and various methods for obtaining the inputs in an Internet environment.

Our main results and conclusions are:

- Placement algorithms should incorporate client workload information, such as client distance and request rate, in their placement decisions. Such algorithms consistently perform a factor of 2 – 5 better than a workload-oblivious random algorithm.

- A greedy algorithm that places replicas based upon both a distance metric and request load performs the best (i.e., its median performance is within a factor of 1.1 – 1.5 of optimal). A hot spot algorithm based upon request load only performs nearly as well (its median performance is within a factor of 1.6 – 2 of optimal). A tree-based algorithm developed for proxy cache hierarchies [85] performs better than random placement, but not as well as the algorithms for general graph topologies. The reason that the tree-based algorithm performs inferior to the heuristics is that the tree-based assumptions prune possibly better placement choices. The optimal in the pruned search space can be significantly worse than a good solution in the entire search space, as found by the heuristics.

- The placement algorithms are not very sensitive to noise in the estimates of distance and load used as inputs to the algorithms. Even with rough estimates of client distance and request load salted with random noise, the algorithms perform nearly as well as when they used perfect knowledge. For example, when the salted error is as high as a factor of 4, the greedy algorithm stays within a factor of 2 of the super-optimal in most cases.

- When deployed, the placement algorithms must predict future request load based upon past information. We show that the algorithms can use a simple moving window average for predicting load with negligible impact on performance.

- The relative performance of the placement algorithms is consistent across network topologies (tree, random graph, AS hop-count), topology parameters (the number of nodes, inter-node distance), trace workloads, and noise in the inputs.

Based upon our results, we conclude that a greedy algorithm for Web server replica placement can provide content distribution networks with performance that is close to optimal. Although the greedy algorithm depends upon estimates of client distance and load predictions, we find that it is relatively insensitive to errors in these estimates and is therefore a viable and practical algorithm for use in the general Internet environment where workload information will likely be imperfect.

In the next chapter, we investigate a transport-based approach to improving the efficiency of the Web.

Chapter 5

Optimizing TCP Performance for Web Transfers

Web data transfers are short and bursty [60, 123, 86]. Such a workload interacts poorly with TCP, the dominant transport protocol in today's Internet.

In this chapter, we propose a new technique, which we call TCP/SPAND, to effectively eliminate the slow start penalty for short data transfers. In TCP/SPAND, network performance information is shared among many co-located hosts to estimate each connection's fair share of the network resources. Based on such estimation and the file transfer size, the TCP sender determines the *optimal* initial congestion window size at connection start up and upon restart after an idle period. Instead of doing the slow start, it uses a pacing scheme to smoothly send out the packets in its initial window. Once all the packets in the initial window have been paced out, the sender switches back to the behavior of standard TCP.

We implement TCP/SPAND in the `ns` simulator [100] and use extensive simulations to evaluate its performance. Our results show that TCP/SPAND significantly reduces latency for short transfers even in presence of multiple heavily congested bottlenecks. Meanwhile, TCP/SPAND is TCP-friendly. The significant performance improvement is not at the expense of degrading performance of the connections using standard TCP [71]. Therefore, deploying TCP/SPAND at Web servers can provide great performance benefits. Furthermore, our system is designed to be incrementally deployable in today's Internet. It only involves modifications at the server side. All the client side applications can be left untouched.

The rest of the chapter is organized as follows. In Section 5.1, we analytically derive the optimal initial congestion window as a function of certain network path characteristics and the transfer size. Section 5.2 presents the design and implementation of TCP/SPAND, an incrementally deployable architecture that allows us to apply our analytical results to today's Internet. Section 5.3 presents simulation results to evaluate the effectiveness of our approach. We summarize in Section 5.4.

5.1 Choosing Optimal Initial Congestion Window

In this section, we show how to determine the optimal initial congestion window, denoted as $cwnd_{opt}$, that minimizes the completion time for a data transfer given the transfer size and the network path characteristics. We first analytically derive $cwnd_{opt}$ for a simple scenario in which the network path characteristics remain unchanged (in Section 5.1.1). We then extend our results to more general scenarios by introducing a technique called the *shift optimization* (in Section 5.1.2).

5.1.1 Derivation of $cwnd_{opt}$ for a Simple Scenario

We consider a simple network model in which there is only one TCP connection, the network path characteristics remain unchanged for the duration of the connection, and all losses are due to network congestion.

We use the following notations in our derivations:

- Let W_{opt} = *propagation delay * bottleneck bandwidth.*
- Let $W_c = W_{opt} + B$, where B is the buffer size at the bottleneck router.

W_{opt} is the number of packets the link can hold on its own, while W_c is the total number of packets that the link and the buffer together can hold.

As we know, the slow start algorithm is designed to probe the available network capacity. The throughput during the slow start is very low. More specifically, without delayed ACKs, no more than $2^k - 1$ packets can be sent in k roundtrip times (RTTs) during the slow start; with delayed ACK, the throughput is even lower because the congestion window grows more slowly. Therefore, if the network condition is known and stable, we should try to avoid the slow start, and enter congestion avoidance directly. This can be achieved by setting the initial slow start threshold (*ssthresh*) to be no more than the initial congestion window (*cwnd*). Therefore, we only need to consider the congestion avoidance phase in order to find $cwnd_{opt}$. Moreover, we only consider the *Reno-style* congestion avoidance because it is the dominant TCP flavor in today's Internet.

Consider a TCP/Reno connection that starts at time 0 with an initial *cwnd* of $\lfloor \frac{W_c+1}{2} \rfloor$ and has an infinite amount of data to send. Define $S(t)$ as the service rate it receives at time t, and $W(t)$ as its congestion window size at t. Figure 5.1 depicts roughly how $S(t)$ and $W(t)$ evolve with time, going through periodical *epochs*. A

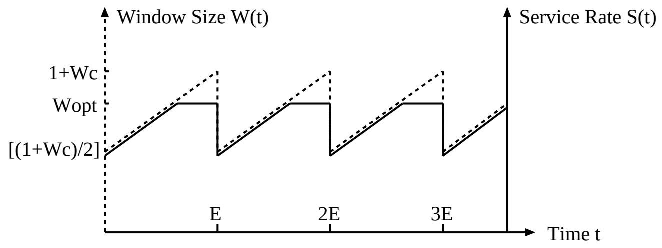


Figure 5.1: Evolution of service rate and congestion window during congestion avoidance for TCP/Reno. E is the duration of an epoch.

similar figure also appeared in [65]. (For simplicity, we ignore the time required by the fast retransmission algorithm to wait for 3 duplicated ACKs.)

Let

E: the duration of an epoch

F_E: the number of packets sent during an epoch

b: the average number of packets acknowledged by an ACK

Close inspection of the congestion window evolution allows us to estimate F_E: During each epoch, the congestion window size starts from $\lfloor \frac{W_c+1}{2} \rfloor$ and increases linearly in time, with a slope of $\frac{1}{b}$ packets per roundtrip time. When the window size reaches $W_c + 1$, the sender injects $W_c + 1$ packets into the network. Since the network can hold at most W_c packets, the last packet will get dropped. After sending another W_c packets (corresponding to the W_c ACKs for those packets that are not dropped), the sender detects the loss by 3 duplicated ACKs and recovers from it through the fast retransmission. Then the window drops back to $\lfloor \frac{W_c+1}{2} \rfloor$ and a new epoch begins. Therefore, the total number of packets sent during each epoch is:

$$F_E = b * (\sum_{x=\lfloor \frac{W_c+1}{2} \rfloor}^{W_c} x) + 2 * W_c + 1 \tag{5.1}$$

For a given transfer with size F, choosing $cwnd_{opt}$ is essentially the problem of fitting a block with size F in the $S(t)$ curve to minimize the transfer time. More precisely, we want to find $s \in [0, E)$ to minimize T, which satisfies $\int_s^{s+T} S(t)dt = F$. (We can then choose $W(s)$ as our $cwnd_{opt}$.) For this problem, we have the following theorem:

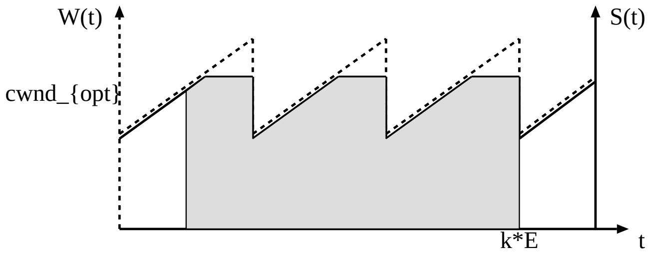


Figure 5.2: Minimize the completion time by having the transfer end at some epoch boundary.

Theorem 1 *As illustrated in Figure 5.2, we can minimize the completion time for a given transfer with size F by choosing the initial cwnd appropriately so that the transfer ends at an epoch boundary, that is, at time $k * E$, for some integer k.*

The detailed proof for Theorem 1 can be found in Appendix A. Below we give some intuition behind the theorem.

Consider a transfer that starts at time $s \in [0, E)$ with a completion time of $T \leq E$. Clearly, there are only two cases: (i) $s + T \leq E$, and (ii) $s + T > E$. As illustrated in Figure 5.3, by comparing the area of the shaded regions, it is evident that in both cases the amount of data transferred during interval $[E - T, E]$ is no less than the data transferred in $[s, s + T]$, which is the transfer size. This means that if we can have the original transfer end at time E, the completion time does not increase.

Based on Theorem 1 we can further derive the value of $cwnd_{opt}$. The result is summarized in the following theorem:

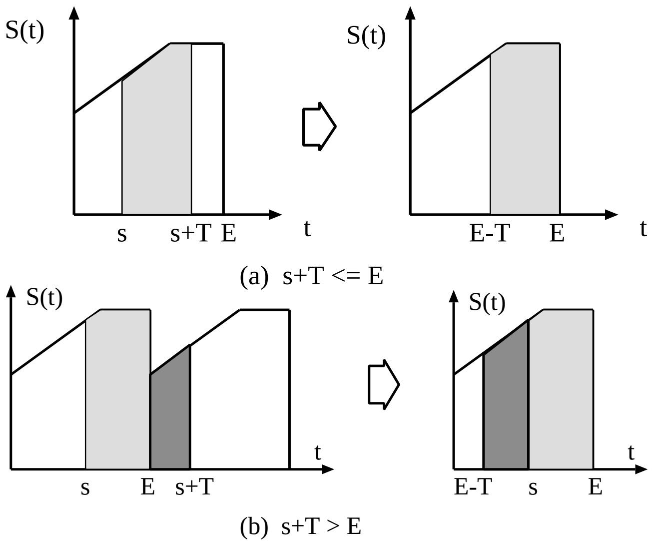


Figure 5.3: Fixing the width of the shaded region to $T \leq E$, we can maximize the area of the region by having the region end at E.

Theorem 2 *The optimal initial cwnd ($cwnd_{opt}$) that minimizes the completion time of a given file transfer with size F can be determined as the largest integer m ($\leq W_c + 1$) satisfying*

$$b * \sum_{i=m}^{W_c} i + (2 * W_c + 1) \geq F \bmod F_E \tag{5.2}$$

where F_E is defined in Equation 5.1.

The derivation of Theorem 2 is given in Appendix B.

Note that the $cwnd_{opt}$ in Theorem 2 is derived for the case in which the transfer ends at some epoch boundary. But ending at an epoch boundary means the connection has to experience a loss during its final epoch. Although in theory, the loss can be detected and recovered quickly through the fast retransmission, in reality it may still be desirable to avoid such an additional loss. This can be achieved by having the transfer end with its *cwnd* equal W_c instead of $W_c + 1$. Accordingly, the initial *cwnd* becomes the largest integer m ($\leq W_c$) satisfying

$$b * \sum_{i=m}^{W_c} i \geq F \bmod F_E \tag{5.3}$$

5.1.2 Shift Optimization

In the previous section, we showed that $cwnd_{opt}$ given by Theorem 2 minimizes the transfer time for a given file, assuming that W_c remains unchanged throughout the duration of the transfer.

However since it is not realistic to assume a constant value of W_c, we relax this assumption through a technique which we call the *shift optimization*. It is based on the following insight. We can often reduce $cwnd_{opt}$ given in Theorem 2, while still keeping the same integer number of RTTs spent in the transfer. In this way, we can achieve close to the minimum transfer time (The transfer time after shift optimization is not more than one roundtrip time longer than the minimum transfer time.), but with a smaller and thus *safer* initial window. More importantly, as we will demonstrate later in this section, the shift optimization makes our algorithm less sensitive to the exact value of W_c.

The shift optimization works as follows. If $cwnd_{opt}$ determined by Equation 5.3 satisfies $b * (\sum_{i=cwnd_{opt}}^{W_c} i) > (F \text{mod} F_E)$, we can reduce the initial *cwnd* without increasing the integer number of required roundtrips by shifting the entire transfer in Figure 5.2 towards left.

The exact amount of reduction in the *cwnd* can be determined as follows. Suppose we reduce the initial *cwnd* by Δ. Since it takes roughly $b*(W_c - cwnd_{opt} + 1)$ roundtrips for *cwnd* to grow from $(cwnd_{opt} - \Delta)$ to $(W_c - \Delta)$, reducing the initial *cwnd* by Δ leads to reduction in the total amount of transferred data by $\Delta * (b * (W_c - cwnd_{opt} + 1))$ during the same time interval. In order to keep the same integer number of required roundtrips, such reduction should be no more than $b * \sum_{i=cwnd_{opt}}^{W_c} i - F \bmod F_E$. This gives us

$$\Delta = \lfloor \frac{b * \sum_{i=cwnd_{opt}}^{W_c} i - F \bmod F_E}{b * (W_c - cwnd_{opt} + 1)} \rfloor \tag{5.4}$$

A simple example of the shift optimization is illustrated in Figure 5.4. In this example, we have $b = 1$, $W_c = 10$, and transfer size $F = 11$. From Equation 5.3, we obtain $cwnd_{opt} = 9$, which results in a completion time of 2 RTTs (more precisely, more than 1 but less than 2 full RTTs). After the shift optimization, we get a much smaller initial *cwnd* of 5, while the completion time becomes 2 full RTTs, which is only slightly larger than that before the shift optimization.

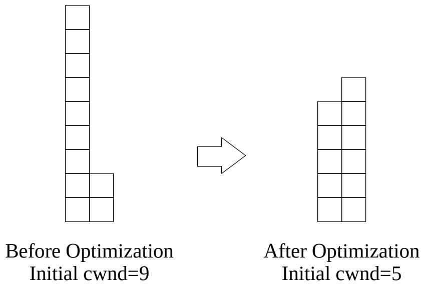
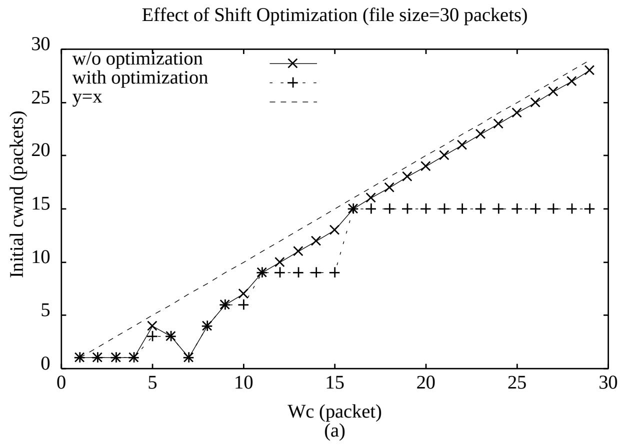


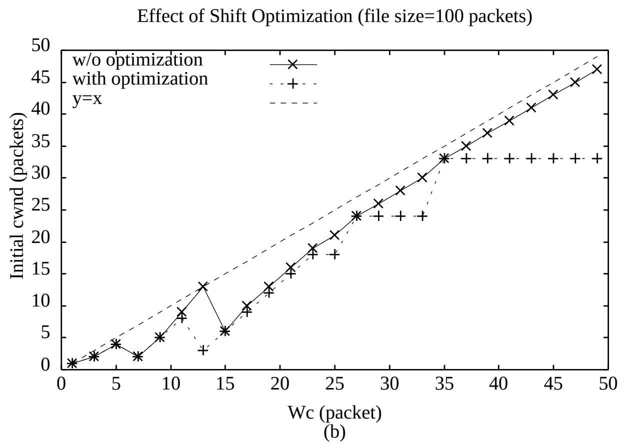
Figure 5.4: A simple example of the *shift optimization*. ($b = 1$, $W_c = 10$, and transfer size $F = 11$.)

To further demonstrate the effect of the shift optimization, we keep the transfer size to be either 30 or 100 packets, and plot the computed initial *cwnd* with and without the shift optimization as a function of W_c. The results are summarized in Figure 5.5.

We make two observations: First, for both transfer sizes, with the shift optimization, the initial *cwnd* is more conservative than directly reusing W_c (corresponding to curve $y = x$ in the figure). Second, with the shift optimization, the computed *cwnd* is insensitive to the value of W_c, especially when W_c is large. Multiple values of W_c can result in the same initial *cwnd*. Such insensitivity allows us



(a)



(b)

Figure 5.5: The effect of the *shift optimization*.

to compute a near optimal initial *cwnd* even if W_c varies over time or if an accurate estimate for W_c is not available.

We have demonstrated that the shift optimization makes the choice of the initial *cwnd* both conservative and insensitive to the value of W_c. Meanwhile, the latency is only increased by a small amount (less than one RTT). In the rest of this chapter, we will use $cwnd_{opt}$ to refer to the optimal initial *cwnd* after the shift optimization.

5.1.3 Shared Networks

So far we have only considered non-shared networks. We can extend our results to shared networks by redefining W_c as a connection's *share* of the available network resources, which can be estimated as one segment smaller than the congestion window size before the TCP sender detects a loss during congestion avoidance. Studies of Internet traces show that the WAN performance is reasonably stable over terms of several minutes [109, 17, 116]; meanwhile, *nearby* hosts experience similar or identical throughput performance within a time period measured in minutes [109, 17, 116]. Such level of stability suggests sharing performance information both temporarily and spatially (across many co-located hosts) can help to more accurately determine network performance, in particular, a connection's *fair share* of network resources (W_c).

5.2 TCP/SPAND: System Design and Implementation

In this section, we present the design and implementation of TCP/SPAND, a system that allows applications like Web servers or FTP servers to avoid the slow start penalty by applying the theory we developed in Section 5.1. Our design only involves modifications to the server side. All the client applications can be left untouched. This makes TCP/SPAND incrementally deployable in today's Internet.

5.2.1 System Architecture

Our design of TCP/SPAND, similar to SPAND [116, 115], uses a *performance gateway* that monitors all traffic entering and leaving an organization's network. For each destination network, the gateway gathers network performance information from all the active TCP flows to that destination. It then aggregates such information to estimate the current network state.

The gateway needs to track two types of performance information, which are described below:

- W_c, which reflects the network resources available to a TCP flow. It is required for choosing the optimal initial *cwnd* as shown in Section 5.1. W_c can be estimated as one segment smaller than the current congestion window size when a TCP sender detects a loss during congestion avoidance. For a short data transfer, it is possible that an entire transfer completes without experiencing any loss. In this case, its W_c can be estimated as the congestion window size when the connection terminates.

- The *round-trip time* information. It is needed to determine the initial rate for smoothly sending out the packets in the initial congestion window, which we will discuss in detail in Section 5.2.2. It is also useful for determining the initial TCP timeout value.

At connection start-up or upon restart after an idle period, the application sitting on top of a TCP sender (e.g., a Web server) extracts the current estimates of W_c and RTT from the performance gateway. It then computes $cwnd_{opt}$ as described in Section 5.1 based on such estimates, as well as the transfer size, which is locally available. It then uses a `setsockopt` system call to initialize the parameters such as the *cwnd* and *srtt* for the underlying TCP sender. Instead of doing the slow start, the TCP sender directly enters congestion avoidance. (This can be achieved by setting *ssthresh* to be no more than *cwnd*). Meanwhile, the TCP sender uses a pacing scheme to smoothly send out the packets in the initial window. Once all the packets in the initial window have been paced out, it switches back to the behavior of standard TCP.

5.2.2 Implementation Issues

There are a number of questions we need to answer in order to actually implement TCP/SPAND:

- What is the right scope for information sharing and aggregation? That is, among which TCP flows should the performance information be shared and aggregated?

- How does the performance gateway collect performance information for active TCP flows?

- After collecting the performance information, what algorithms should the performance gateway use to aggregate the performance information and estimate the current network state?

- How can the applications sitting on top of TCP extract the current estimates of W_c and RTT from the performance gateway?

- What pacing scheme should the TCP senders use to send out packets in the initial congestion window? How can such a scheme be implemented?

In the remainder of this section, we discuss the potential solutions as well as different implementation strategies for these problems in turn.

Determining the Scope for Sharing and Aggregation

First, we discuss how to determine the right scope for information sharing and aggregation.

Ideally, we would like to share performance information among flows that share the same bottleneck router at the same time. Unfortunately, in the current Internet architecture, it is very difficult to determine which network flows share the same bottleneck, because there is no easy way to determine where a packet was dropped.

To get around such difficulty, we decide to use conservative approximations based on the destination IP address. Two possible approximations have been proposed in [115]: *host locality*, that is, flows that share a common destination IP address; and *network locality*, that is, flows that share the same destination network. We choose to use the latter, which allows more sharing.

Note that it is somewhat difficult to determine the network address from an IP address because the length of the network address can vary. We use a simple heuristic which assumes that IP addresses sharing the most significant 24 bits belong to the same network [115].

Below we evaluate the accuracy of the 24-bit heuristic using access logs recorded at the MSNBC news site [96]. Our traces are from busy hours (from 9:00am to noon) on three consecutive weekdays (from Tuesday, August 03 1999 to Wednesday, August 05 1999). They consist of 10,688,728 HTTP requests to MSNBC, with requests for inline images excluded.

For each client IP address, we use reverse DNS lookup to resolve its host name, and take the last two segments of the host name as its domain name. We then report how often client IP addresses sharing the most significant 24 bits get resolved to different domain names. Altogether there are 656,559 IP addresses in the access logs, among which 475,803 (72.48%) can be successfully resolved to host names via reverse DNS lookup. Of those 475,803 IP addresses, there are 106,669 distinct 24-bit subnet addresses, and only 6715 (6.3%) subnet addresses contain IP addresses that are resolved to different domain names. This demonstrates the high accuracy of the 24-bit heuristic.

Collecting Performance Information

The second important problem is how the performance gateway collects performance information from active TCP flows.

There are several possible implementation strategies for this problem:

First, the TCP senders can record the performance information as socket state variables. The application sitting on top of TCP can periodically get such information using the `getsockopt` system call and then report to the performance gateway by sending a special *performance report* packet. This is similar to the approach used in the original SPAND system.

Alternatively, it is possible to modify TCP so that a TCP sender piggybacks the performance information in its normal outbound data packets by introducing a new TCP option. When the performance gateway captures such packets, it can extract the performance information it needs. The TCP receivers can simply ignore this option. The bandwidth and processing overhead for the new TCP option is unlikely to cause a performance concern because the performance information does not have to be reported very frequently. However, the TCP senders may need to negotiate the option in advance in order to interoperate with existing implementation of TCP receivers [70]. A similar approach for piggybacking the performance information is to steal bits from normal IP headers [120]. This approach does not require any TCP option negotiation and can be implemented with only sender-side modifications.

As a third alternative, the performance gateway itself can infer the performance information by passively monitoring all the traffic entering and leaving the organization's network and then reconstructing the TCP protocol state. This is similar to [87]. Compared to the first two approaches, the passive approach has the advantage that it does not consume any extra bandwidth and does not require any modification to the sender's protocol stack.

Information Aggregation

After collecting the performance information, the performance gateway needs to aggregate such information to estimate the current network state accurately.

Currently, we use a simple sliding window averaging algorithm to aggregate the performance information. The performance gateway keeps a sliding window of S minutes in duration. It uses the average of all values in the past S minutes as the estimates for the current W_c and RTT.

In case there is not enough performance information in the sliding window, the performance gateway simply informs the TCP senders to do the slow start. As part of our future research, we are interested in exploring the possibility of exponentially decaying the estimation for W_c in this case.

The only control parameter in our algorithm is S, the size of the sliding window. The choice of S involves some tradeoffs: On one hand, we want S to be as large as possible in order to maximize sharing; on the other hand, a large S means the performance gateway needs to keep a large amount of state; in addition, the choice of S needs to match the level of network stability.

Currently, we set S to 5 minutes. It clearly matches the level of stability reported in [109, 17, 116], which is a few minutes. Below we use MSNBC traces to demonstrate that a 5-minute sliding window can achieve significant sharing while only requiring moderate amount of state kept by the performance gateway.

Figure 5.6 plots the cumulative distribution of the time gap between two consecutive requests from the same client network. (We use the 24-bit heuristic described earlier to determine the client network address.) As we can see, around 90% of cases, we get another request from the same client network within 5 minutes. This suggests that using a 5-minute sliding window, most Web transfers are able to benefit from sharing the previous network performance information.

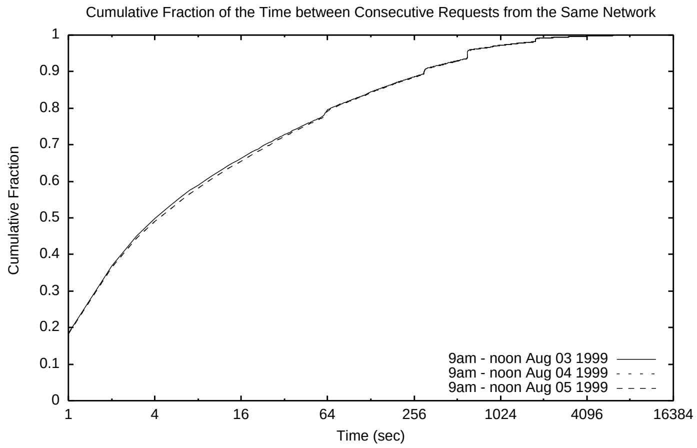


Figure 5.6: Cumulative distribution of the time between two consecutive requests from the same network. The 24-bit heuristic is used to determine the network address from an IP address.

We also assess the amount of state the performance gateway needs to keep. Since the performance gateway keeps state for each destination network, the total amount of required state is proportional to the number of different destination networks showing up in a 5-minute interval. For each request in the MSNBC traces, we count the number of different client networks that appeared in the next 5 minutes. We then plot the cumulative distribution for all these numbers in Figure 5.7. From the figure it is evident that the performance gateway only needs to keep state for 15,000 to 25,000 different destination networks even during the busiest periods. This can be easily handled by modern computers.

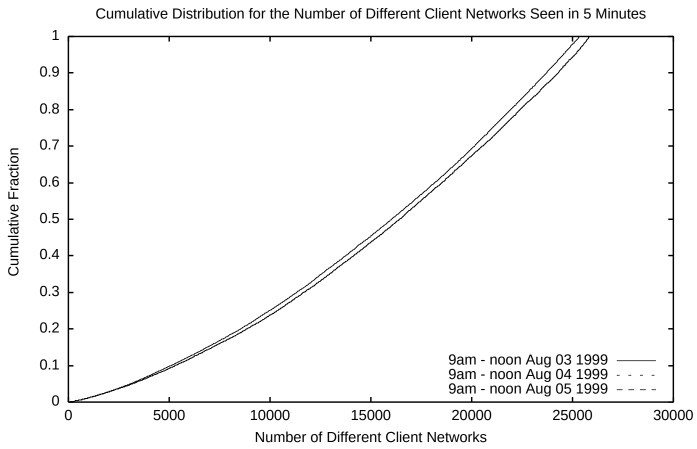


Figure 5.7: Cumulative distribution of the number of different client networks seen in 5 minutes.

Retrieving the Current Estimate of Network State

The next problem to discuss is how the application sitting on top of TCP can retrieve the current estimate of W_c and RTT from the performance gateway.

There are at least two solutions to this problem:

First, similar to SPAND, the application can explicitly query for such information by sending a special *performance inquiry* packet to the performance gateway.

Alternatively, when the performance gateway infers that some application needs the current estimates of W_c and RTT, it can actively *push* such information to the host on which the application runs.

There are several simple heuristics that can be used by the performance gateway to determine if an application needs such estimates. For example, when the performance gateway captures an incoming file transfer request (e.g., an HTTP GET request), or even simpler, when it captures an inbound packet for a new TCP connection, or for a connection that has been idle for a while, it can safely infer that the corresponding TCP flow needs the estimates. These heuristics can be implemented easily, especially when the gateway already uses the Windmill approach [87] to collect performance information through passive monitoring and protocol reconstruction.

Pacing

Lastly, we discuss how to send out the packets in the initial congestion window. In TCP/SPAND, the TCP senders do not go through the slow start procedure after initializing *cwnd*. However, the $cwnd_{opt}$ derived in Section 5.1 can be potentially large. Sending out all the packets in the initial congestion window back to back is clearly unacceptable because it can result in a large burst overflowing the network bottleneck buffer.

A natural solution to this problem is to have the TCP sender use a pacing scheme to smoothly send out all the packets in the initial window. Once all packets in the initial window have been sent, the sender can switch back to the behavior of standard TCP.

There are several pacing schemes proposed in the literature [14, 103, 129]. Here we introduce an alternative scheme based on the leaky-bucket. In this scheme, a TCP sender uses a leaky bucket (more specifically, a token bucket) to shape its outgoing traffic. When the sender has a packet to send and the *cwnd* permits, it checks the token bucket. If there are sufficient tokens, the packet is sent immediately; otherwise, it is delayed using a fine-grained timer until there are enough tokens. The depth of the token bucket can be configured to limit the maximum burstiness of the outgoing traffic. It is set to 4 segments in our simulations. The token filling rate is set to $\frac{cwnd}{srtt}$ so that the average sending rate is no more than *cwnd* packets per roundtrip.

In order to implement the leaky-bucket based pacing scheme, we need a fine-grained timer to reschedule a segment for later transmission in case there are no sufficient tokens. Meanwhile, since the token filling rate is inversely proportional to *srtt*, we need a relatively accurate estimate of RTT, which cannot be achieved with the 200 ms or 500 ms timer granularity in standard TCP. In our simulations, we use a 50 ms timer. If the TCP *timestamp* option is available, it can be used to improve the accuracy of RTT estimation further.

There is evidence to suggest that the overhead of software timers is not likely to be significant with modern processor technologies. ([56] reports an overhead of the order of a few microseconds.) Moreover, the timer overhead is unlikely to be a significant addition to the cost of taking interrupts and processing ACKs that goes with ACK clocking [103].

5.2.3 Implementation Status

Currently, we have implemented TCP/SPAND in the `ns` network simulator [100]. Our implementation is based on TCP NewReno [67, 58], a variant of TCP that uses partial new ACK information to recover from multiple packet losses in a window at the rate of one packet per RTT. As described above, the performance gateway uses a 5-minute sliding window to aggregate the performance information. For simplicity, we assume that the communication between the performance gateway and the other hosts in the organization's network is instantaneous. This is reasonable, because compared to the large delay in a WAN connection, the communication latency in a LAN environment is negligible.

5.3 Simulation Results

In this section, we use extensive simulations in the `ns` network simulator to study the performance of TCP/SPAND. We first discuss our simulation topology and experiment methodology, and then present detailed results for various simulation scenarios.

5.3.1 Simulation Topology

In our simulations, we use single-bottleneck topologies to uncover and illuminate the important issues, and more realistic multiple-bottleneck topologies to evaluate the performance of TCP/SPAND in *real-world* scenarios.

The single-bottleneck topology is shown in Figure 5.8. One or more bursty connections are established between a subset of the sources on the left and sinks on the right. The bottleneck buffer is 10 KB. The bottleneck router uses FIFO scheduling and drop-tail buffer management. All non-bottleneck links have 10 Mbps capacity and 1 ms one-way propagation delay. We consider three scenarios shown in Table 5.1.

The multiple-bottleneck topology is illustrated in Figure 5.9. In this topology, a set of M user flows traverse a congested network path that consists of K hops. Cross traffic is generated at each intermediate router R_i ($i = 1, 2, \ldots, K$) from C

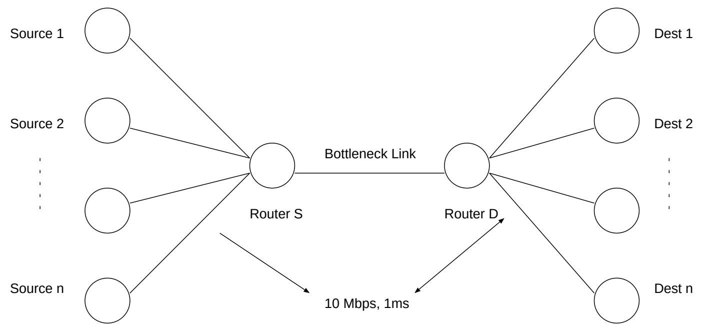


Figure 5.8: Single-bottleneck topology. The bottleneck buffer is 10 KB. The settings for bottleneck link are summarized in Table 5.1.

Table 5.1: Different simulation scenarios for the single-bottleneck topology.

Scenario	Bandwidth	Link Delay	Descriptions
1	1.6 Mbps	50 ms	typical terrestrial WAN links with close to T1 bandwidth
2	1.6 Mbps	200 ms	typical geostationary satellite links with close to T1 bandwidth
3	45 Mbps	200 ms	typical geostationary satellite links with T3 bandwidth

cross-traffic sources. A similar topology has been recently used in [46]. Each router has a 10 KB buffer and uses FIFO scheduling and drop-tail buffer management. All links other than those between adjacent routers have 10 Mbps capacity and 1 ms one-way propagation delay. As for the links between adjacent routers, we consider two scenarios summarized in Table 5.2, which roughly correspond to Scenario 1 and 2 for the single-bottleneck topology. (Note that the aggregated buffer size is larger in the multiple-bottleneck scenario.)

Table 5.2: Different simulation scenarios for the multiple-bottleneck topology. K is the number of hops, where $K \geq 2$.

Scenario	Bandwidth	Link Delay	Descriptions
4	1.6 Mbps	$50/(K-1)$ ms	roundtrip time (RTT) similar to Scenario 1
5	1.6 Mbps	$200/(K-1)$ ms	roundtrip time (RTT) similar to Scenario 2

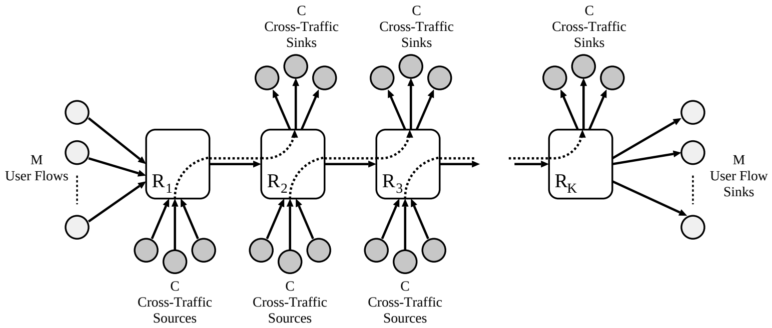


Figure 5.9: The multiple-bottleneck topology. The bottleneck buffer is 10 KB. The settings for the bottleneck link are summarized in Table 5.2.

5.3.2 Experiment Methodology

We set up our experiments in the following way to mimic the behavior of Web transfers. Each experiment consists of 40 rounds. In each round every sender transfers one file with start time uniformly distributed around a central point by a time interval (denoted as *jitter*). Between each round there is 120 seconds of idle time.

We use the average completion time of all file transfers in an experiment as our performance metric. For each simulation configuration, we report the mean of 10 runs of an experiment. We also looked at the variation, but since it is very small compared to the mean, we do not report it here.

We compare the performance of TCP/SPAND with the following four variants of TCP:

- Reno with the slow start restart (reno-ssr): TCP/Reno which enforces the slow start when restarting data flow after an idle period.

- Reno without the slow start restart (reno-nssr): TCP/Reno which reuses the prior congestion window upon restarting after an idle period. (This is the scheme used in SunOS.)

- NewReno with the slow start restart (newreno-ssr): TCP/NewReno with the restart behavior similar to reno-ssr.

- NewReno without the slow start restart (newreno-nssr): TCP/NewReno with the restart behavior similar to reno-nssr.

The maximum window size of all TCP connections in our simulations is set to 100 KB. The TCP segment size is set to 1 KB. Moreover, in order to remove the performance difference due to different timer granularities, all TCP flavors use the same timer granularity of 50 ms unless otherwise specified. Finally, just like in [101, 103], all the TCP protocols in our experiment use one-way connections instead of two-way connections. That is, TCP classes derived from `TcpAgent` are used instead of those derived from `FullTcpAgent` in the `ns` simulator. Consequently, there is no overhead of 3-way handshaking at connection setup. We believe removing such connection setup overhead is necessary for us to better understand the performance impact of the slow start procedure, which is the major focus of TCP/SPAND. Avoiding connection setup overhead is orthogonal to avoiding the slow start penalty and has already been well studied in the literature. For example, P-HTTP [104] can amortize such overhead across multiple transfers effectively.

5.3.3 Performance Evaluation on Single-Bottleneck Topologies

In this section, we evaluate the performance of TCP/SPAND on the single-bottleneck topologies illustrated in Figure 5.8. We examine the multiple-bottleneck topologies in the following section.

Performance evaluation with many concurrent web transfers

First, we compare the performance using five telnet sessions as cross traffic, which help to avoid deterministic behavior. The inter-arrival times for telnet sessions are drawn from the "tcplib" distribution as implemented in `ns`.

Varying the number of competing connections We evaluate the performance as the number of competing TCP connections varies from 1 to 30 while the transfer size is kept at 30 KB, which is the average Web transfer size [86].

Figure 5.10 shows the results for two different scenarios where the bottleneck link is a T1 link (1.6 Mbps) with a latency of 50 ms or 200 ms (Scenario 1 and 2 in Table 5.1). As shown in the figure, TCP/SPAND leads to significant reduction in the average completion time. Compared with reno-ssr and newreno-ssr, TCP/SPAND reduces latency by more than 50% in most cases, which means a 100% improvement in average data rate. Compared to the reno-nssr and newreno-nssr, the completion time reduction is smaller but still significant, about 20% in most cases. Notice that the reno-nssr and newreno-nssr are well-known to be overly aggressive but still perform considerably worse than TCP/SPAND. There are two major reasons for this: first, directly reusing previous *cwnd* is not optimal; second, sending all packets in the initial window at once is usually too bursty and can cause more losses than TCP/SPAND.

Varying the transfer size Now we compare performance as the transfer size varies. As shown in Figure 5.11, TCP/SPAND reduces the completion time over wide range of transfer sizes in both scenarios we study. In percentage terms, the improvement decreases as the transfer size increases. This is what we would expect, because avoiding the slow start penalty has a much bigger impact on small

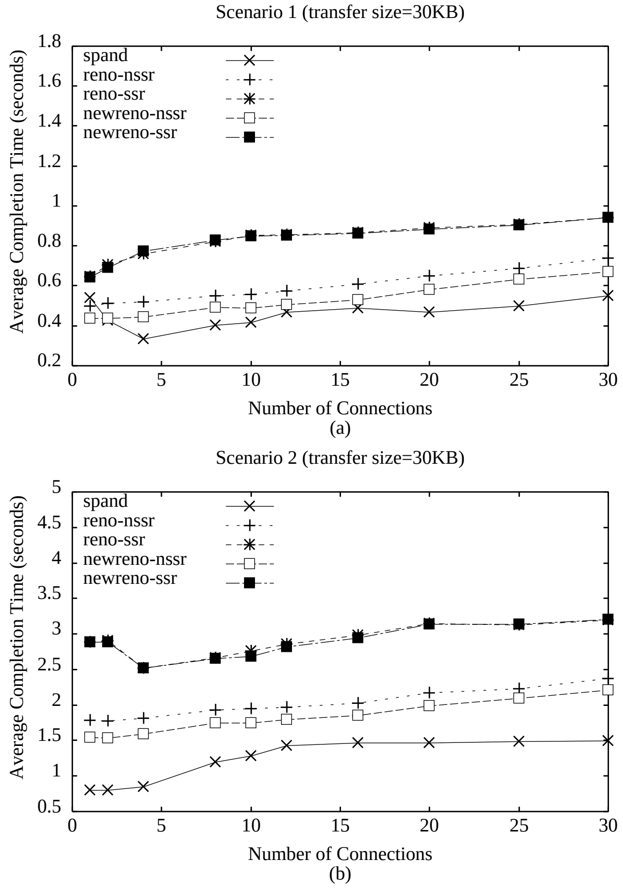


Figure 5.10: Performance comparison for different numbers of connections. The bottleneck link has setting of either Scenario 1 or Scenario 2 (defined in Table 5.1). In each round, file transfers start within 10 seconds (i.e., jitter=10 sec). 5 telnet sessions are used to avoid deterministic behavior.

transfers than on larger ones. The improvement increases with link delay due to the larger bandwidth delay product. Moreover, the improvement tends to decrease as the number of connections increases. This is mainly because as the number of connections increases, each connection's share of W_c decreases, so that the impact of the slow start phase on TCP performance becomes less significant.

Performance evaluation with ON/OFF UDP flows as cross traffic

It has been reported in [107] that WWW-related traffic tends to be self-similar in nature. In [131], it is shown that self-similar traffic may be created by using

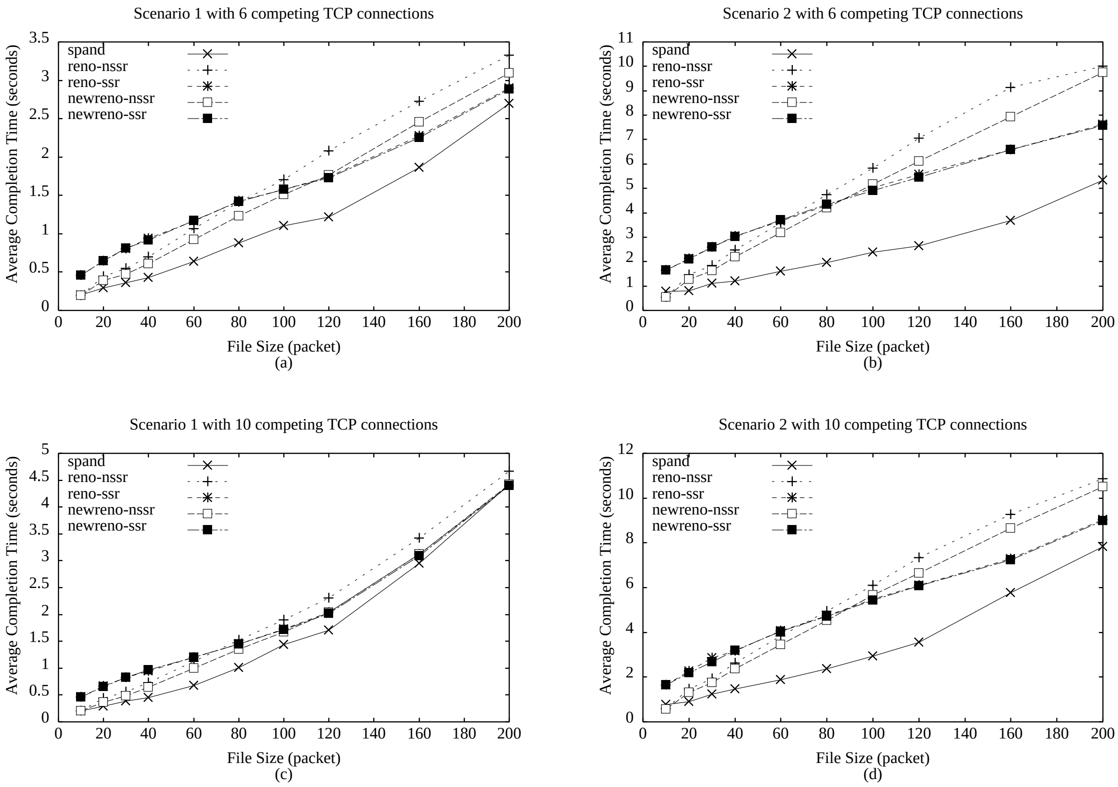
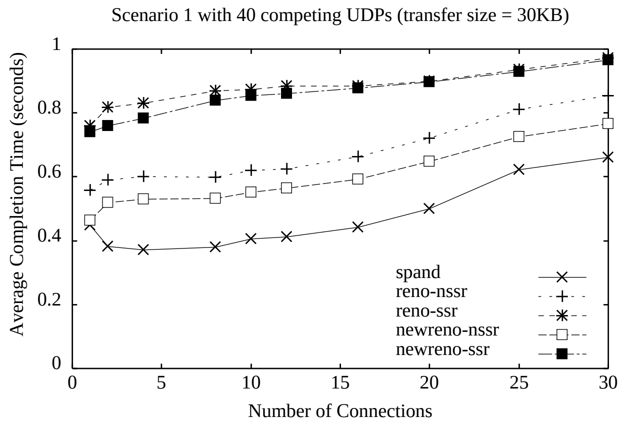


Figure 5.11: Performance comparison for different transfer sizes. Either 6 or 10 TCP connections compete for the bottleneck link with the setting of either Scenario 1 or Scenario 2 (defined in Table 5.1). In each round, file transfers start within 10 seconds (i.e., jitter=10 sec). 5 telnet sessions are used to avoid deterministic behavior.

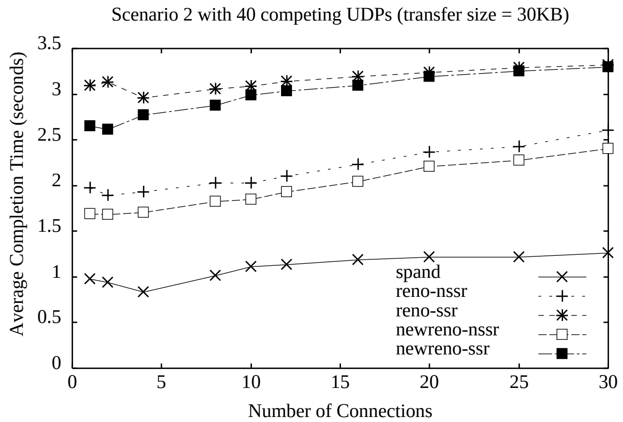
several ON/OFF UDP sources whose ON/OFF times are drawn from heavy-tailed distributions such as the Pareto distribution. In this section, we evaluate the performance of TCP/SPAND with ON/OFF UDP flows as cross traffic.

As before (in Section 5.3.3), we first evaluate performance as the number of competing connections varies while the transfer size is kept at 30 KB. The simulation results are illustrated in Figure 5.12. In both scenarios, TCP/SPAND reduces latency by 35% to 65% compared with reno-ssr and newreno-ssr, which means a 60% to 200% improvement in average data rate. Compared with reno-nssr and newreno-nssr, the latency reduction is around 25% to 50%.

Figure 5.13 shows the result of varying the transfer size and keeping the number



(a)



(b)

Figure 5.12: Performance comparison for different numbers of connections with 40 ON/OFF UDP flows as cross traffic. The ON/OFF times of the UDP sources are drawn from Pareto distributions with the "shape" parameter set to 1.2. The mean ON time is 1 second and the mean OFF time is 2 seconds. During ON times, the sources transmit with a rate of 12 Kbps. In both (a) and (b), the jitter for the transfer start time in each round is 10 seconds.

of connections constant. As we can see, the performance benefit of TCP/SPAND is greater for larger bandwidth.

Performance evaluation with bulk transfers (FTP) as cross traffic

Finally we compare the performance with bulk transfers (FTP) as cross traffic. The simulation results are summarized in Figure 5.14 and Figure 5.15. In all simulations, a number of FTP connections make the bottleneck link heavily loaded. Figure 5.14(a) shows the results of using 2 FTP sessions as cross traffic to create

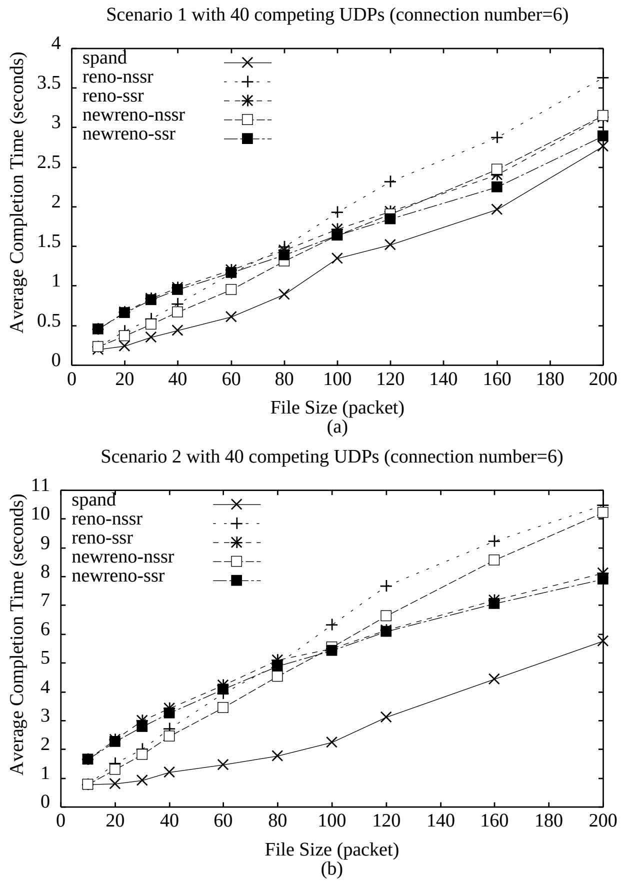


Figure 5.13: Performance comparison for different transfer sizes with UDP as cross traffic. The settings are the same as in Figure 5.12.

contention for the T1 link with 200 ms latency. TCP/SPAND reduces the average completion time by 35% to 50% compared to reno-ssr and newreno-ssr. The improvement is smaller compared to reno-nssr and newreno-nssr, which are known to be aggressive. When the bottleneck link is T3, the performance improvement is more significant, as shown in Figure 5.14(b). Also notice that in this case, the average completion time for TCP/SPAND connections is slightly larger than reno-nssr and newreno-nssr (less than 1 more RTT) when the connection number is smaller than 8. This is mainly because the *shift optimization* and pacing make TCP/SPAND much more conservative. With a small number of short connections, TCP/SPAND cannot fully probe the available bandwidth, which is very large in

Scenario 3. However, when there are long-lived connections, this is not a problem.

Figure 5.15 shows the results of varying the transfer size and keeping the connection number constant. Again, the performance benefit of TCP/SPAND is greater for larger bandwidth.

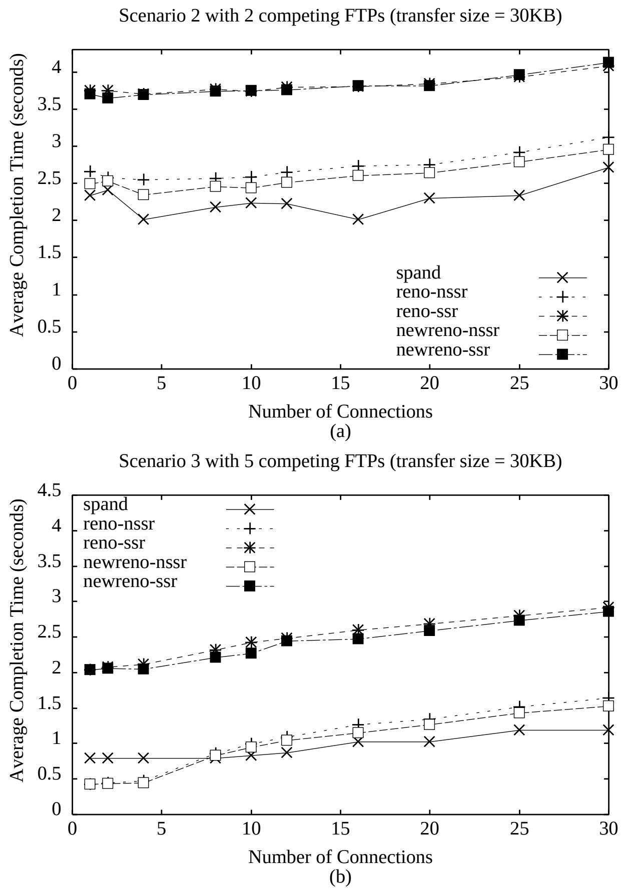


Figure 5.14: Performance comparison for different numbers of connections with FTPs and 5 telnet sessions as cross traffic. In (a), the jitter for the transfer start time in each round is 10 sec; while in (b), the jitter is 0.1 sec.

5.3.4 Performance Evaluation on Multiple-Bottleneck Topologies

In this section, we evaluate the performance of TCP/SPAND when the underlying network path is heavily congested and has multiple bottlenecks.

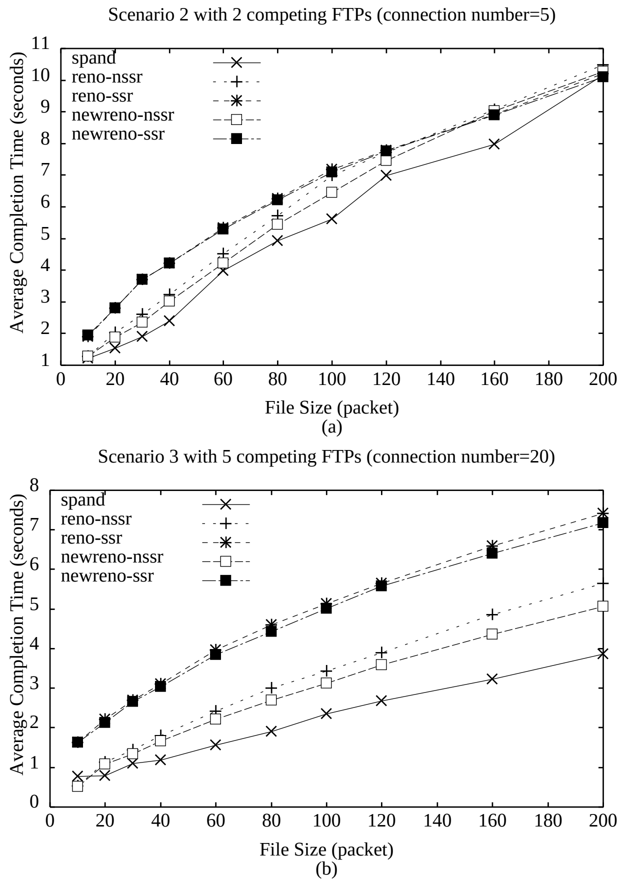


Figure 5.15: Performance comparison for different transfer sizes with FTP as cross traffic. The settings are the same as in Figure 5.14.

We use the multiple-bottleneck topology illustrated in Figure 5.9. The number of hops (K) is fixed to 5. At each intermediate router R_i ($i = 1, 2, 3, 4$), the cross traffic is generated from $C = 40$ ON/OFF UDP connections. Just like in Section 5.3.3, the ON/OFF time of the UDP sources are drawn from Pareto distributions with the "shape" parameters set to 1.2. The mean ON time is 1 second and the mean OFF time is 2 seconds.

Performance evaluation with 12 Kbps UDP Sources

We first evaluate performance when each ON/OFF UDP source generates cross traffic at a rate of 12 Kbps during ON time, which is the same as for the single-bottleneck case in Section 5.3.3.

As before, we first keep the transfer size at 30 KB and vary the number of competing connections. As illustrated in Figure 5.16, for both scenarios we have considered (Scenario 4 and 5), TCP/SPAND leads to significant reduction in the completion time: 30% to 65% over reno-ssr and newreno-ssr, 25% to 55% over reno-nssr and newreno-nssr. Such performance improvement is comparable to the single-bottleneck case shown in Figure 5.12.

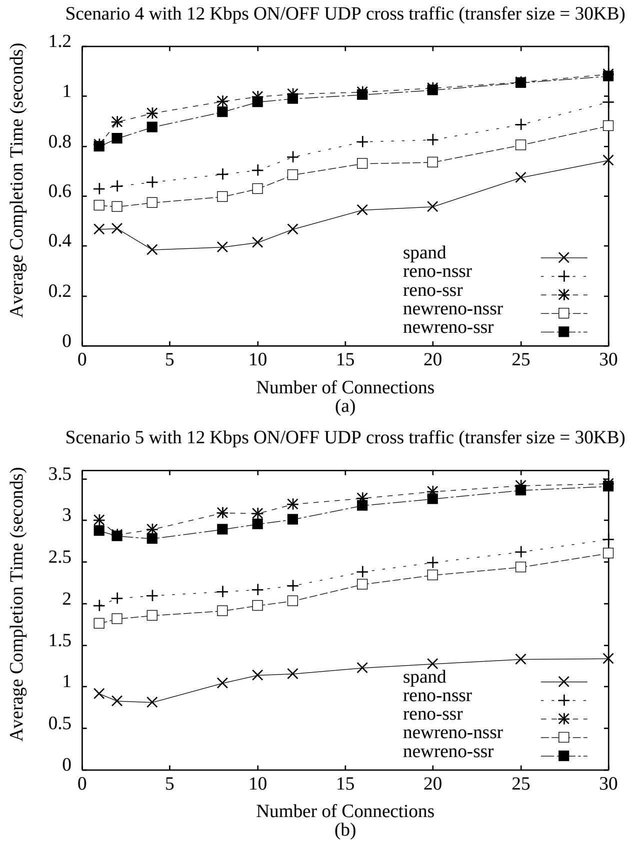


Figure 5.16: Performance comparison for different numbers of connections with 12 Kbps ON/OFF UDP flows as cross traffic. In both (a) and (b), the jitter for the transfer start time in each round is 10 seconds.

We then fix the number of competing connections to 6 and vary the transfer size. It is evident from Figure 5.17 that TCP/SPAND again achieves significant performance improvement similar to the single-bottleneck case shown in Figure 5.13.

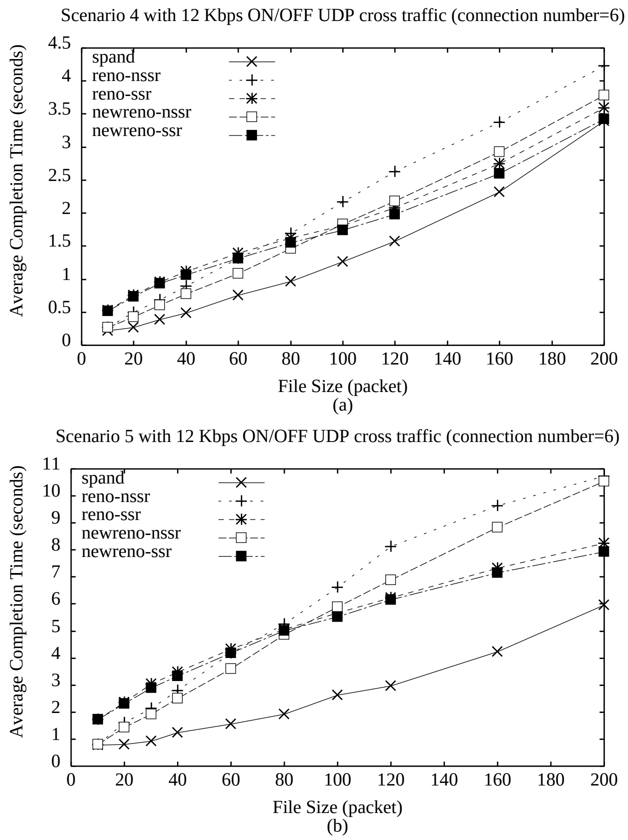
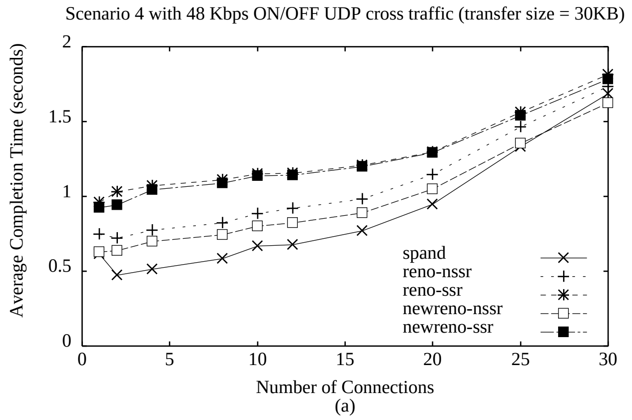


Figure 5.17: Performance comparison for different transfer sizes with 12 Kbps ON/OFF UDP flows as cross traffic. The settings are the same as in Figure 5.16.

Performance evaluation with 48 Kbps UDP Sources

To further investigate the performance of TCP/SPAND under heavy congestion, we increase the sending rate of the UDP sources during ON time to 48 Kbps and re-do all the experiments. The simulation results are summarized in Figure 5.18 and Figure 5.19.

From the significant increase in the completion time, it is evident that the underlying network path is highly congested. But even under such heavy congestion, TCP/SPAND consistently outperforms the other TCP flavors. Of course, the performance improvement decreases. This is not surprising, because the impact of the initial congestion window becomes less significant as the network becomes highly congested.



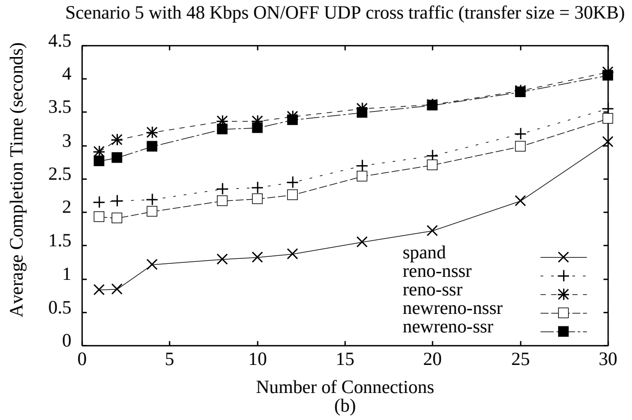


Figure 5.18: Performance comparison for different numbers of connections with 48 Kbps ON/OFF UDP flows as cross traffic. In both (a) and (b), the jitter for the transfer start time in each round is 10 seconds.

5.3.5 TCP-Friendliness

In this section, we demonstrate the performance improvement of TCP/SPAND does not come at the expense of degrading performance of connections using standard TCP. In other words, TCP/SPAND is TCP-friendly.

We show this by considering a mixture of TCPs on the single-bottleneck topology. One half of the connections use TCP/SPAND, while an equal number of connections use reno-ssr, one of the least aggressive TCP schemes. We then compare their performance with the case in which all connections use reno-ssr. The jitter for transfer start time during each round is set to 0.1 second to create maximum contention for the bottleneck bandwidth. Again 5 telnet sessions are introduced

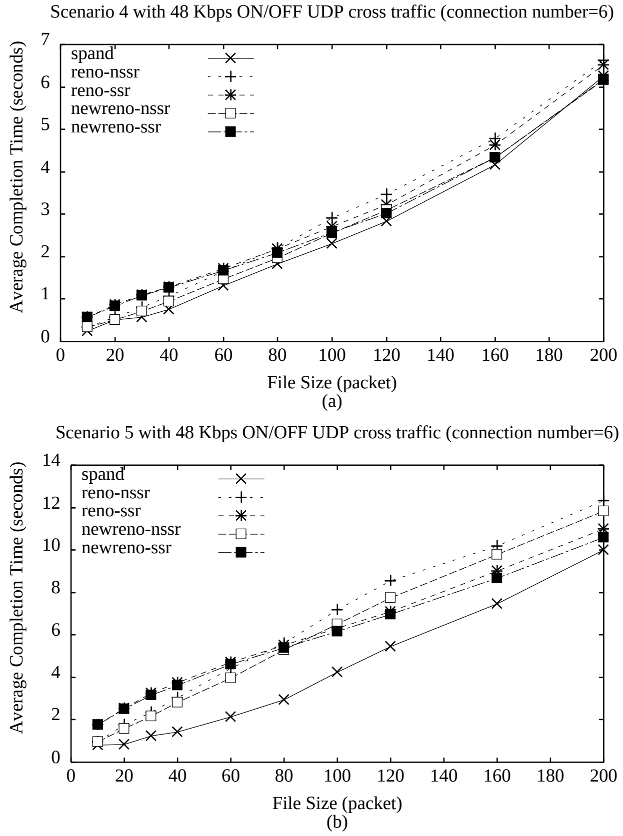


Figure 5.19: Performance comparison for different transfer sizes with 48 Kbps ON/OFF UDP flows as cross traffic. The settings are the same as in Figure 5.18.

to avoid deterministic behavior.

First, we use a 50 ms timer for reno-ssr, which is the same as TCP/SPAND. Figure 5.20 summarizes the simulation results for four different settings, where the bottleneck link is a T1 link with a latency of either 50 ms (Scenario 1) or 200 ms (Scenario 2), and the transfer size is set to at either 30 KB or 100 KB.

In all scenarios we have considered, the performance of reno-ssr, when mixed with TCP/SPAND, is virtually the same as when all connections use reno-ssr. This demonstrates TCP/SPAND is TCP-friendly even under heavy contention.

Also worth mentioning is that in Scenario 1 (as shown in Figure 5.20(a) and (b)), TCP/SPAND performs almost the same as reno-ssr. This is because the jitter is only 0.1 second. In such case, each connection's share of W_c is very small due to heavy contention. Consequently, the optimal initial *cwnd* is very close to 1, which makes TCP/SPAND behave almost the same as reno-ssr. In comparison, the performance benefit of TCP/SPAND is much greater in Scenario 2 (as shown in Figure 5.20(c) and (d)), where the bandwidth delay product is larger.

We also use 200 ms timer granularity for reno-ssr to further evaluate the TCP-friendliness of TCP/SPAND. The results are illustrated in Figure 5.21. In all four settings, TCP/SPAND outperforms reno-ssr. Meanwhile, reno-ssr experiences little or no performance degradation in the presence of TCP/SPAND (less than 5% in most cases). For all cases in which degradation occurs, there are a large number of connections and the network is highly congested. In such cases, reno-ssr frequently needs to use timeout for loss recovery. The 200 ms timer granularity makes such recovery unnecessarily slow and hence degrades the performance. This is a well-known problem of traditional TCP itself.

5.3.6 Summary of Simulation Results

To summarize, in this section we use extensive simulations to evaluate the performance of TCP/SPAND. TCP/SPAND consistently outperforms reno-ssr, newreno-ssr, reno-nssr, and newreno-nssr in all simulation scenarios, even in the presence of multiple heavily congested links. The performance benefit is greatest when each connection's share of W_c is large. In such cases, TCP/SPAND can reduce latency by 35% to 65% compared with reno-ssr and newreno-ssr; or by 20% to 50% compared with reno-nssr and newreno-nssr. Meanwhile, such significant performance improvement does not come at the cost of degrading unenhanced TCP connections. We have demonstrated that TCP/SPAND is TCP-friendly even under heavy contention.

There are three major factors that contribute to the good performance of TCP/SPAND. First, by sharing and aggregating performance information among many co-located hosts, the sender can have a quite accurate estimate of current network conditions. Second, the initial *cwnd* used by TCP/SPAND is chosen based on our theoretical analysis for the optimal initial *cwnd*. We also employ *shift optimization* (described in Section 5.1) to make this choice conservative and insensitive to the accuracy of the estimate of current network characteristics. In this way, TCP/SPAND can utilize available bandwidth efficiently and safely when each connection's share of W_c is large. When the network is heavily loaded, TCP/SPAND tends to be conservative. Particularly, when each connection's share of W_c is less than 4K bytes, the choice of the initial *cwnd* in TCP/SPAND is more conservative than what is proposed in RFC 2414 [5]. Finally, by using a pacing scheme to send out packets in the initial congestion window, TCP/SPAND reduces the burstiness of TCP upon start-up and restart effectively.

5.4 Summary

In this chapter, we investigated the possibility of speeding up short data transfers by avoiding the slow start penalty. By analyzing the TCP start-up dynamics, we derive the optimal initial congestion window ($cwnd_{opt}$) as a function of the transfer size and a connection's share of network resources (W_c). We then propose an incrementally deployable architecture called TCP/SPAND, which accurately estimates a connection's fair share of network resources by sharing performance information among a large number of co-located hosts. Based on the estimates and the transfer size, a TCP sender can further compute the optimal initial congestion window size.

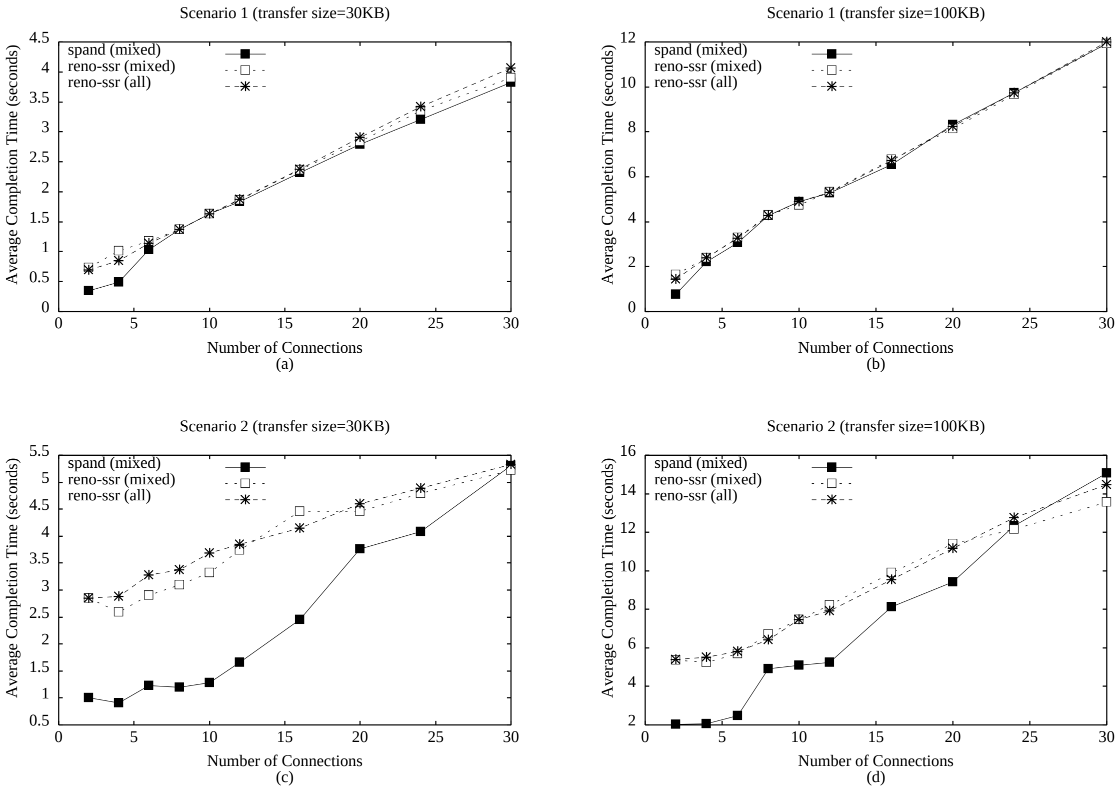


Figure 5.20: TCP friendliness. 5 telnet sessions are used as background traffic. The transfer start time in each round has a jitter of 0.1 sec. Transfer size is either 30 KB or 100 KB. The timer granularity for both reno-ssr and TCP/SPAND is 50 ms.

Instead of doing the slow start, it directly enters congestion avoidance and uses a pacing scheme to smoothly send out the packets in its initial congestion window. We do extensive simulations using `ns` simulator to evaluate the performance of the resulting system. Our results show that TCP/SPAND significantly reduces latency for short transfers even in the presence of multiple heavily congested bottlenecks. Meanwhile, the performance benefit does not come at the expense of degrading the performance of connections using standard TCP.

In the next and final chapter, we summarize the work presented in this thesis, and discuss how this work may be extended in the future.

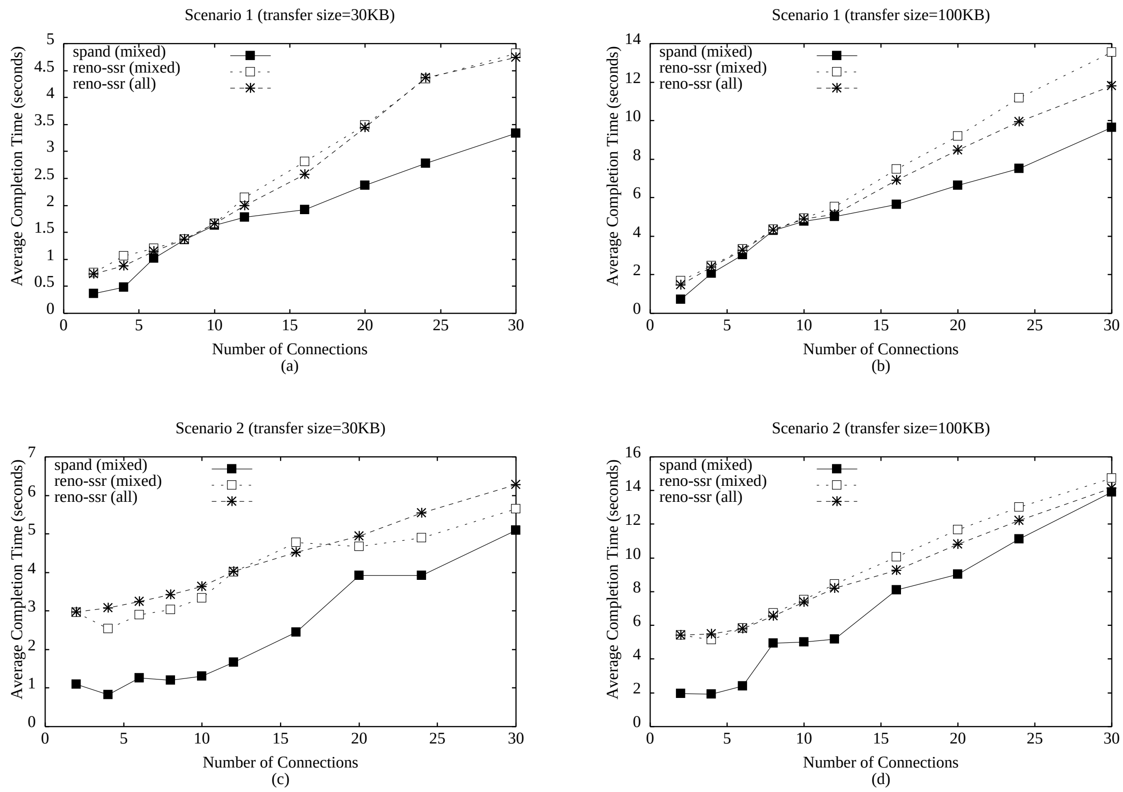


Figure 5.21: TCP friendliness. The configuration is the same as in Figure 5.20 except that reno-ssr uses 200 ms timer granularity instead of 50 ms. TCP/SPAND still uses 50 ms timer granularity.

Chapter 6

Conclusions and Future Work

In this section, we summarize our contributions, and point out directions for future work, both in addressing the limitations of this work, and in exploring new avenues.

6.1 Summary

This thesis identifies three major challenges in providing efficient Web services:

1. The workload of the World Wide Web is still not well understood. In particular, few studies look at the content dynamics and access dynamics at popular Web sites.

2. Designing a scalable hosting services infrastructure is an open issue. Most work so far has concentrated on Web caching to enhance Web performance. Hosting services has emerged as a new service paradigm to address the issues that Web caching encounters. There are many new research issues that need to be addressed before such an architecture can be fully exploited. In particular, how to properly provision the hosting servers in the Internet to

enhance users' performance is a very important research problem, but has received little attention.

3. Web transfers are short and bursty. This is very different from bulk transfers (e.g., ftp), for which TCP was originally designed. The poor interaction between Web transfers and TCP results in unnecessarily low throughput and high latency.

We address these challenges as follows.

1. We study both the content and access dynamics of the MSNBC Web site, one of the busiest sites in the Internet today. Our findings have significant implications for the design of efficient Web services.

2. We develop several algorithms to efficiently provision the content distribution networks. Our evaluation shows that a greedy algorithm for solving the Web server replica placement problem can provide content distribution networks with performance that is close to optimal. Although the greedy algorithm depends upon estimates of client distance and load predictions, we find that it is relatively insensitive to errors in these estimates and therefore is a viable algorithm for use in practice.

3. We propose TCP/SPAND to optimize TCP startup performance. Our results show that TCP/SPAND significantly reduces latency for short transfers even in the presence of multiple heavily congested bottlenecks. Meanwhile, its performance gain is not at the expense of degrading the performance of non-enhanced TCP.

6.2 Directions for Future Work

There are several interesting directions based on the work described in this thesis. Some of them are extensions of our work, while others explore new avenues.

6.2.1 Further Understanding of the Web Workload

A limitation of our study is that it is difficult to determine how well the results derived from the MSNBC site generalize to other large sites in the Internet. While there may certainly be characteristics that are peculiar to the MSNBC site, we believe that the operation of the site is quite typical of large news sites such as CNN and ABCNews. Many of our findings and implications are likely to be applicable to other news sites. It will be very useful to study other news sites to confirm this.

Furthermore, there is a wide variety of commercial Web servers besides news sites. These include e-commerce servers (e.g., online bookstores), query servers (e.g., search engines), upload servers (e.g., sites where people submit their tax returns), and streaming servers (e.g., Internet radio stations). Different types of Web servers may have very different workload characteristics, and hence may require different techniques for performance optimization. Therefore, a fruitful approach to better understanding the Web workload would be to study the workload of different types of Web servers.

6.2.2 Improving the Efficiency of Placement Algorithms

We developed several algorithms that can make the Web server placement decision automatically. An interesting next step is to explore incremental versions of the placement algorithms that take into account the cost of changing the set of replica sites. Ideally, for placement strategies with similar performance, we prefer the one that incurs the least amount of perturbation to the system. For example, if site A is already hosting a Web service, then we prefer not to replace it with another replica site unless the performance degradation of continuing to use A is significant. It is also useful to study distributed versions of the placement algorithms to further improve the scalability of the system. One possible distributed algorithm would be to make the algorithm hierarchical. That is, we first run the placement algorithm on the higher levels in the hierarchy to decide how many replicas each region at lower levels in the hierarchy should have. Then each region at the lower levels runs the placement algorithm independently to decide the placement strategy for its region.

6.2.3 Server Selection & Redirection Techniques

An orthogonal issue to providing high-performance Web hosting services is to develop efficient server selection and redirection mechanisms. There have been significant research efforts in this area, such as [10, 50, 43, 54, 49, 75, 23, 74]. However, almost all the techniques were developed for delivering static objects, such as images. The rapid growing multimedia traffic makes it necessary to develop effective server selection and redirection schemes to handle multimedia traffic.

6.2.4 Further Investigation of TCP/SPAND

We proposed the TCP/SPAND technique to optimize TCP startup performance for short Web transfers. A number of avenues for future work remain. First, it will be very useful to implement TCP/SPAND in real systems and evaluate its performance through Internet experiments. Meanwhile, it is also interesting to further evaluate TCP/SPAND in a controlled environment by running more simulations. In particular, increasing deployment of active queue management makes it important to understand the performance impact of different router dropping policies (e.g. RED) on TCP/SPAND.

Second, further work is needed in developing effective techniques for determining which network flows share a bottleneck. Despite the recent progress made in [114], it still remains an open problem to find a solution that scales to many flows over a variety of traffic conditions.

Finally, we need to investigate better algorithms for information aggregation. In particular, it is worthwhile to examine techniques that exponentially decay the estimated W_c when there is little or no performance information available for the active connections. Recently some techniques have been proposed to address similar problems [16, 64]. Another interesting direction is to investigate the performance benefits of sharing congestion information across heterogeneous data streams as proposed in [102].

Appendix A

Proof for Theorem 1

Proof: Suppose $cwnd^* = W(t^*)$ ($t^* \in [0, E]$) achieves the minimum completion time of T^*. We have:

$$\int_{t^*}^{t^*+T^*} S(t)dt = F \tag{A.1}$$

Let $n = \lfloor T^*/E \rfloor$ and $T_r = T^* - n * E \in [0, E)$. We take two steps to prove Theorem 1: We first prove the following equation (Equation A.2), then we demonstrate how Equation A.2 leads to Theorem 1.

$$\int_{E-T_r}^{E} S(t)dt \geq \int_{t^*}^{t^*+T_r} S(t)dt \tag{A.2}$$

Step 1: Proof for Equation A.2. We only need to consider two cases:

1. $t^* + T_r \in [0, E]$ (illustrated in Figure A.1(a)). In this case Equation A.2 follows naturally from the monotonicity of $S(t)$ over $[0, E)$.

2. $t^* + T_r \in (E, 2E]$ (illustrated in Figure A.1(b)). Since $S(t)$ is monotonically increasing over $[0, E)$, we have

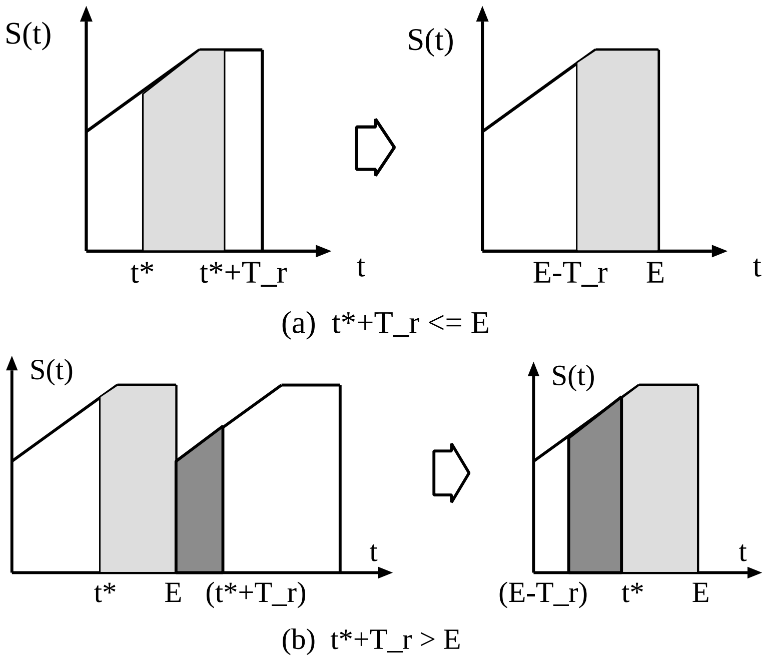


Figure A.1: Two cases for Equation A.2

$$\int_{E-T_r}^{t^*} S(t)dt \geq \int_0^{t^*-(E-T_r)} S(t)dt \tag{A.3}$$

Consequently,

$$\begin{aligned}
\int_{E-T_r}^{E} S(t)dt &= \int_{E-T_r}^{t^*} S(t)dt + \int_{t^*}^{E} S(t)dt \\
&\geq \int_0^{t^*-(E-T_r)} S(t)dt + \int_{t^*}^{E} S(t)dt \quad (\Leftarrow \text{Eqn A.3}) \\
&= \int_{0+E}^{t^*-(E-T_r)+E} S(t)dt + \int_{t^*}^{E} S(t)dt \\
&= \int_{E}^{t^*+T_r} S(t)dt + \int_{t^*}^{E} S(t)dt \\
&= \int_{t^*}^{t^*+T_r} S(t)dt
\end{aligned}$$

This completes the proof for Equation A.2.

Step 2: From Equation A.2 to Theorem 1. From the periodicity of $S(t)$ and Equation A.2, we have

$$
\begin{aligned}
&\int_{(E+n*E)-T^*}^{E+n*E} S(t)dt = \int_{E-T_r}^{E+n*E} S(t)dt \\
= \;& \int_{E-T_r}^{E} S(t)dt + \int_{E}^{E+n*E} S(t)dt \\
= \;& \int_{E-T_r}^{E} S(t)dt + \int_{t^*+T_r}^{(t^*+T_r)+n*E} S(t)dt \\
\geq \;& \int_{t^*}^{t^*+T_r} S(t)dt + \int_{t^*+T_r}^{(t^*+T_r)+n*E} S(t)dt \quad (\Leftarrow \text{Eqn A.2}) \\
= \;& \int_{t^*}^{t^*+T^*} S(t)dt = F
\end{aligned}
$$

This means we can transfer a file with size $\geq F$ during period $[(E + n * E) - T^*, E + n * E]$. In other words, if the given transfer with size F ends at an epoch boundary $E + n * E$, the completion time is no more than T^*. But T^* is already optimal, therefore the completion time should be exactly T^*. That is,

$$
\int_{(E+n*E)-T^*}^{E+n*E} S(t)dt = F \tag{A.4}
$$

This completes our proof for Theorem 1.

Appendix B

Derivation of $cwnd_{opt}$ given in Theorem 2

Based on Theorem 1, we can derive $cwnd_{opt} = W(E + n * E - T^*)$ as follows.

We first calculate the integer n. Since $n = \lfloor T^*/E \rfloor$, we have: $0 < E + n * E - T^* \leq E$. Therefore,

$$\begin{aligned} n * F_E = \int_E^{E+n*E} S(t)dt &\leq \int_{(E+n*E)-T^*}^{E+n*E} S(t)dt \\ &< \int_0^{E+n*E} S(t)dt = (1+n) * F_E \end{aligned}$$

That is, $n * F_E \leq F < (1+n) * F_E$. Since n is an integer, we immediately have $n = \lfloor F/F_E \rfloor$. Consequently,

$$\begin{aligned} \int_{(E+n*E)-T^*}^{E} S(t)dt &= \int_{(E+n*E)-T^*}^{E+n*E} S(t)dt - \int_E^{E+n*E} S(t)dt \\ &= F - n * F_E = F - \lfloor F/F_E \rfloor * F_E \end{aligned}$$

That is,

$$\int_{(E+n*E)-T^*}^{E} S(t)dt = F \bmod F_E \tag{B.1}$$

In the same way as we derive F_E in Equation 5.1, we can get

$$\int_{(E+n*E)-T^*}^{E} S(t)dt \approx b * \sum_{i=cwnd_{opt}}^{W_c} i + (2 * W_c + 1) \tag{B.2}$$

From Equation B.1 and B.2, we immediately get Theorem 2.

Bibliography

[1] http://www.abcnews.com.

[2] A. Aggarwal, S. Savage, and T. Anderson. Understanding the Performance of TCP Pacing. *Proc. of IEEE INFOCOM '2000*, Tel-Aviv, Israel, March 2000.

[3] http://www.akamai.com.

[4] M. Allman, S. Dawkins, D. Glover, J. Griner, T. Henderson, J. Heidemann, H. Kruse, S. Ostermann, K. Scott, J. Semke, J. Touch, and D. Tran. Ongoing TCP Research Related to Satellites. Internet Draft draft-ietf-tcpsat-res-issues-05.txt, November 1998.

[5] M. Allman, S. Floyd, and C. Partridge. Increasing TCP's Initial Window. RFC-2414, September 1998.

[6] M. Allman, D. Glover, and L. Sanchez. Enhancing TCP Over Satellite Channels using Standard Mechanisms. RFC-2488, September 1998.

[7] M. Allman and C. Hayes. An Evaluation of TCP with Larger Initial Windows. *ACM Computer Communication Review*, Vol. 28, No. 3, July 1998.

[8] V. F. Almeida, M. G. Cesorio, R. C. Fonseca, W. Meira Jr., and C. D. Murta. The Influence of Geographical and Cultural Issues on the Cache Proxy Server Workload. In *Proc. of the World Wide Web Conference*, Brisbane, Australia, April 1998.

[9] V. Almeida, A. Bestavros, M. Crovella, and A. Oliveira. Characterizing Reference Locality in the WWW. In *Proc. of PDIS '96*, December 1996.

[10] D. Andresen, T. Yang, V. Holmedahl, and O. H. Ibarra. SWEB: Towards a scalable World Wide Web server on multicomputers. In *Proc. of the 10th Int. Parallel Processing Symp. (IPPS '96)*, Honolulu, Hawaii, USA, April 1996.

[11] M. F. Arlitt and C. L. Williamson. Web Server Workload Characterization: The Search for Invariants. In *Proc. of SIGMETRICS '96*, Philadelphia, PA, USA, May 1996.

[12] M. F. Arlitt and C. L. Williamson. Internet Web Servers: Workload Characterization and Performance Implications. IEEE Transactions on Networking, Vol. 5, No. 5, October 1997.

[13] M. F. Arlitt, and T. Jin. Workload Characterization of the 1998 World Cup Web Site. IEEE Network, Vol. 14, No. 3, pp. 30-37, May/June 2000.

[14] H. Balakrishnan, V. Padmanabhan, and R. Katz. The Effects of Asymmetry on TCP Performance. *Proc. of ACM/IEEE Mobicom '97*, Budapest, Hungary, September 1997.

[15] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, M. Stemm, and R. H. Katz. TCP Behavior of a Busy Web Server: Analysis and Improvements. In *Proc. of INFOCOM '98*, San Francisco, CA, USA, March 1998.

[16] H. Balakrishnan, H. Rahul, and S. Seshan. An Integrated Congestion Management Architecture for Internet Hosts. *Proc. of SIGCOMM '99*, Boston, MA, USA, September 1999.

[17] H. Balakrishnan, S. Seshan, M. Stemm, and R. Katz. Analyzing Stability in Wide-Area Network Performance. *Proc. of SIGMETRICS '97*, Seattle, WA, USA, June 1997.

[18] P. Barford, A. Bestavros, A. Bradley, and M. E. Crovella. Changes in Web Client Access Patterns: Characteristics and Caching Implications. World Wide Web, Special Issue on Characterization and Performance Evaluation, Vol. 2, pp. 15-28, 1999.

[19] T. Berners-Lee, R. Fielding, and H. Frystyk. *Hypertext Transfer Protocol - HTTP/1.0*. RFC-1945, May 1996.

[20] R. Braden. Extending TCP for Transactions - Concepts. RFC-1379, November 1992.

[21] R. Braden. T/TCP - TCP Extensions for Transactions Functional Specification. RFC-1644, July 1994.

[22] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web Caching and Zipf-like Distributions: Evidence and Implications. In *Proc. of INFOCOMM '99*, New York City, NY, USA, March 1999.

[23] T. Brisco. DNS support for load balancing. RFC-1794. April 1995.

on Protocols For High-Speed Networks, August 1999.

[24] K. Calvert and E. Zegura. GT Internetwork Topology Models (GT-ITM). http://www.cc.gatech.edu/fac/Ellen.Zegura/gt-itm.

[25] P. Cao and C. Liu. Maintaining Strong Cache Consistency in the World-Wide Web. *Proc. of the 17th International Conference on Distributed Computing Systems*. May 1997. IEEE Transactions on Computers (April 1998) vol. 47, no. 4, page 445-57.

[26] P. Cao, J. Zhang, and K. Beach. Active Cache: Caching Dynamic Contents on the Web. In *Proc. of Middleware '98*, The Lake District, England, 1998.

[27] G. Chandranmenon, and G. Varghese. Reducing Web Latencies Using Reference Point Caching. To appear in *IEEE INFOCOM'2001*, Anchorage, AK, USA, April 2001.

[28] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrel. A Hierarchical Internet Object Cache. In *Proc. of USENIX '96*, San Diego, CA, USA, January 1996.

[29] M. Charikar and S. Guha. Improved Combinatorial Algorithms for the Facility Location and K-Median Problems. In *Proc. of the 40th Annual IEEE Conference on Foundations of Computer Science*, 1999.

[30] M. Charikar, S. Guha, E. Tardos, and D.B. Shmoys. "A constant-factor approximation algorithm for the k-median problem". *Proc. of the 31st Annual ACM Symposium on Theory of Computing (1999)*, Atlanta, GA, USA, May 1999.

[31] F. Chudak. Improved approximation algorithms for uncapacitated facility location. In *Proc. of the 6th IPCO Conference*, pages 180 - 194, Houston, TX, USA, June 1998.

[32] F.A. Chudak and D.B. Shmoys. "Improved approximation algorithms for the capacitated facility location problem". In *Proc. of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (1999)*, S875-S876, 1999.

[33] F. Chudak and D. B. Shmoys. Improved approximation algorithms for the uncapacitated facility location problem. In preparation.

[34] D. D. Clark, and D. L. Tennenhouse. Architectural Considerations for a New Generation of Protocols. *In Proc. of SIGCOMM '90*, Philadelphia, PA, September 1990.

[35] http://ita.ee.lbl.gov/html/contrib/ClarkNet-HTTP.html.

[36] http://www.cnn.com.

[37] K. G. Coffman and A. M. Odlyzko. Internet growth: Is there a “Moore’s Law” for data traffic?

[38] E. Cohen, and H. Kaplan. Prefetching the means for document transfer: A new approach for reducing Web latency. In *Proc. of INFOCOM ’2000*, Tel Aviv, Israel, March 2000.

[39] E. Cohen, B. Krishnamurthy, and J. Rexford. Improving End-to-End Performance of the Web Using Server Volumes and Proxy Filters. In *Proc. of SIGCOMM ’98*, Vancouver, BC, Canada, September 1998.

[40] G. Cornuelos, M. L. Fisher, and G. L. Nemhauser. Location of Bank Accounts to Optimize Float: An Analytic Study of Exact and Approximate Algorithms. Management Sciences, Vol. 23, 1977, pp. 789-810.

[41] C. Cunha, A. Bestavros, and M. Crovella. Characteristics of WWW client-based traces. *Technical Report TR-95-010, Boston University, Computer Science Dept., USA*, April 1995.

[42] A. Demers, S. Keshav, and S. Shenker. Analysis and Simulation of a Fair Queueing Algorithm. Journal of Internetworking Research and Experience, V1, N1, pp 3-26, September 1990; also in *Proc. of ACM SIGCOMM’89*, pp 1-12, Austin, TX, USA, September 1989.

[43] D. M. Dias, K. Kish, R. Mukherjee, and R. Tewari. A scalable and highly available Web server. In *Proc. of the 41st IEEE Computer Society Int. Conf.*, Febuary 1996.

[44] http://www.digitalisland.com.

[45] F. Douglis, A. Feldman, B. Krishnamurthy, and J. C. Mogul. Rate of Change and Other Metrics: A Live Study of the World Wide Web. In *Proc. of USITS ’97*, Monterey, CA, USA, December 1997.

[46] C. Dovrolis, D. Stiliadis, and P. Ramanathan. Proportional Differentiated Services: Delay Differentiation and Packet Scheduling. *Proc. of SIGCOMM ’99*, Boston, MA, USA, September 1999.

[47] B. M. Duska, D. Marwood, and M. J. Feeley. The Measured Access of World Wide Web Proxy Caches. In *Proc. of USITS ’97*, Monterey, CA, USA, December 1997.

[48] V. Duvvuri, P. Shenoy, and R. Tewari. Adaptive Leases: A Strong Consistency Mechanism for the World Wide Web. In *Proc. of INFOCOM ’2000*, Tel-Aviv, Israel, March 2000.

[49] S. G. Dykes, C. L. Jeffery, and K. A. Robbins. An Empirical Evaluation of Client-Side Server Selection Algorithms. In *Proc. of INFOCOM '2000*, Tel-Aviv, Israel, March 2000.

[50] R. S. Engelschall. Apache 1.3 URL Rewriting Guide. http://www.apache.org/docs/misc/rewriteguide.html.

[51] http://www.exodus.com.

[52] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: A scalable wide-area Web cache sharing protocol. In *Proc. of ACM SIGCOMM '98*, Vancouver, BC, Canada, August 1998.

[53] L. Fan, Q. Jacobson, P. Cao, and W. Lin. Web Prefetching Between Low-Bandwidth Clients and Proxies: Potential and Performance. In *Proc. of SIGMETRICS '99*, Atlanta, GA, USA, May 1999.

[54] Z. Fei, S. Bhattacharjee, E. W. Zegura, and M. H. Ammar. A novel server selection technique for improving the response time of a replicated service. In *Proc. of INFOCOM '98*, San Francisco, CA, USA, March 1998.

[55] A. Feldmann, R. Caceres, F. Douglis, and M. Rabinovich. Performance of Web Proxy Caching in Heterogeneous Bandwidth Environments. In *Proc. of INFOCOM '99*, New York City, NY, USA, March 1999.

[56] W. Feng, D. Kandlur, D. Saha, and K. Shin. Understanding TCP Dynamics in an Integrated Services Internet. *Proc. of NOSSDAV '97*, St. Louis, MO, USA, May 1997.

[57] S. Floyd and K. Fall. Promoting the Use of End-to-End Congestion Control in the Internet. *IEEE/ACM Transactions on Networking*, August 1999.

[58] S. Floyd and T. Henderson. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC-2582, Experimental, April 1999.

[59] S. Glassman. A Caching Relay for the World Wide Web. In *Proc. of the First International Conference on the World Wide Web*, CERN, Geneva, Switzerland, May 1994.

[60] S. D. Gribble and E. A. Brewer. System Design Issues for Internet Middleware Services: Deductions from a Large Client Trace. In *Proc. of USITS '97*, Monterey, CA, USA, December 1997.

[61] S. Guha and S. Khuller. Greedy strikes back: improved facility location algorithms. In *Proc. of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 649 - 657, San Francisco, CA, USA, January 1998.

[62] R. Gupta, M. Chen, S. McCanne, and J. Walrand. Receiver-Driven Transport Protocol for the Web. In *Proc. of INFORMS '2000*, San Antonio, TX, November 2000.

[63] J. Gwertzman and M. Seltzer. World-Wide Web Cache Consistency. In *Proc. of USENIX '96*, San Diego, CA, USA, January 1996.

[64] M. Handley, J. Padhye, and S. Floyd. TCP Congestion Window Validation. Technical Report 99-77, Department of Computer Science, University of Massachusetts, Amherst, 1999.

[65] D. Heyman, T. Lakshman, and A. Neidhardt, "A New Method for Analyzing Feedback-Based Protocols with Applications to Engineering Web Traffic over the Internet," *Proc. of SIGMETRICS '97*, Seattle, WA, USA, June 1997.

[66] D. S. Hochbaum. Heuristics for the fixed cost median problem. Math Programming, 22: 148 - 162, 1982.

[67] J. Hoe. Improving the Start-up Behavior of a Congestion Control Scheme for TCP. *Proc. of SIGCOMM '96*, Stanford University, CA, USA, August 1996.

[68] J. Hunt, K. P. Vo, and W. Tichy. Delta Algorithms: An Empirical Analysis. In *ACM Transactions on Software Engineering and Methodology*, Vol. 7, 1998.

[69] IPMA Project. http://www.merit.edu/ipma/.

[70] V. Jacobson, R. Braden, and D. Borman. TCP Extensions for High Performance. RFC-1323, May 1992.

[71] V. Jacobson and M. Karels. Congestion Avoidance and Control. *Proc. of SIGCOMM '88*, Stanford University, CA, August 1988.

[72] K. Jain and V. Vazirani. Primal-Dual Approximation Algorithms for Metric Facility Location and K-Median Problems. Manuscript, March 1999. Available at http://www.cc.gatech.edu/fac/Vijay.Vazirani/k-median.ps.

[73] S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang. On the Placement of Internet Instrumentation. In *Proc. of INFOCOM '2000*, Tel-Aviv, Israel, March 2000.

[74] D. Karger, A. Sherman, A. Berkheimer, B. Bogstad, R. Dhanidina, K. Iwamoto, B. Kim, L. Matkins, and Y. Yerushalmi. Web Caching with Consistent Hashing. In *Proc. of the Eighth International World Wide Web Conference*, Toronto, Canada, May 1999.

[75] E. D. Katz, M. Butler, and R. McGrath. A scalable HTTP server: the NCSA prototype. Computer Networks and ISDN Systems, vol. 27, pp. 155-164, 1994.

[76] S. Keshav. A Control-Theoretic Approach to Flow Control. *Proc. of SIGCOMM '91*, Zurich, Switerland, September 1991.

[77] S. Keshav. Packet-Pair Flow Control. IEEE/ACM Transactions on Networking, February 1995.

[78] S. Keshav. An Engineering Approach to Computer Networking, January 1997.

[79] B. Krishnamurthy and J. Wang. On Network-Aware Clustering of Web Clients. In *Proc. of SIGCOMM '2000*, Stockholm, Sweden, August - September 2000.

[80] B. Krishnamurthy and C. E. Wills. Study of piggyback cache validation for proxy caches in the World Wide Web. In *Proc. of the 1st USENIX Symposium on Internet Technologies and Systems*, Monterey, CA, USA, December 1997.

[81] P. Krishnan and B. Sugla. Utility of co-operating Web proxy caches. In *Proc. of the 7th Int. World Wide Web Conf.*, Brisbane, Australia, April 1998.

[82] T. Lakshman and U. Madhow. Performance Analysis of Window-Based Flow Control using TCP/IP: the Effect of High Bandwidth-Delay Products and Random Loss. *IFIP Transactions C-26, High Performance Networking V*, pp. 135-150, North-Holland, 1994.

[83] W. LeFebvre and K. Craig. Rapid Reverse DNS Lookups for Web Servers. In *Proc. of USITS '99*, Boulder, CO, USA, October 1999.

[84] E. Levy-Abegnoli, A. Iyengar, J. Song, and D. Dias. Design and performance of Web server accelerator. In Proc. of IEEE INFOCOM '99, New York City, NY, USA, March 1999.

[85] B. Li, M. J. Golin, G. F. Ialiano, and X. Deng. On the Optimal Placement of Web Proxies in the Internet. In *Proc. of INFOCOM '99*, New York City, NY, USA, March 1999.

[86] B. Mah. An Empirical Model of HTTP Network Traffic. *Proc. of INFOCOM '97*, Kobe, Japan, April 1997.

[87] G. Malan and F. Jahanian. An Extensible Probe Architecture for Network Protocol Performance Measurement. *Proc. of SIGCOMM '98*, Vancouver, BC, Canada, September 1998.

[88] S. Manley and M. Seltzer. Web Facts and Fantasy. *Proc. of USITS '97*, Monterey, CA, USA, December 1997.

[89] P. R. McManus. A Passive System for Server Selection within Mirrored Resource Environments Using AS Path Length Heuristics. http://proximate.appliedtheory.com/proximate_study.html.

[90] http://www.mediametrix.com.

[91] J. M. Menaud, V. Issarny, and M. Banatre. A new protocol for efficient transversal Web caching. In *Proc. of the 12th Int. Symp. on Distributed Computing*, pages 288-302, September 1998.

[92] S. Michel, K. Nguyen, A. Rosenstein, L. Zhang, S. Floyd, and V. Jacobson. Adaptive Web caching: towards a new caching architecture. Computer Network and ISDN Systems, November 1998.

[93] Microsoft Corporation. http://www.microsoft.com.

[94] J. C. Mogul. Network Behavior of a Busy Web Server and its Clients. *Research Report 95/5, Compaq Western Research Lab*, October 1995.

[95] J. C. Mogul, F. Douglis, A. Feldman, and B. Krishnamurthy. Potential Benefits of Delta Encoding and Data Compression for HTTP. In *Proc. of SIGCOMM '97*, Cannes, French Riviera, France, September 1997.

[96] http://www.msnbc.com.

[97] http://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html.

[98] N. Nishikawa, T. Hosokawa, Y. Mori, K. Yoshidab, and H. Tsujia. Memory-based Architecture for Distributed WWW Caching Proxy. In *Proc. of the 7th WWW Conference*, Brisbane, Australia, April 1998.

[99] A compendium of NP optimization problems. http://www.nada.kth.se/~viggo/problemlist/compendium.html.

[100] UCB/LBNL/VINT Network Simulator - `ns` (version 2). http://www-mash.cs.berkeley.edu/ns, 1997.

[101] V. Padmanabhan. Addressing the Challenges of Web Data Transport. Ph.D. Thesis, UC Berkeley, 1998.

[102] V. Padmanabhan. Coordinated Congestion Management and Bandwidth Sharing for Heterogeneous Data Streams. *Proc. of NOSSDAV '99*, Basking Ridge, NJ, USA, June 1999.

[103] V. Padmanabhan and R. Katz. TCP Fast Start: A Technique for Speeding Up Web Transfers. *Proc. of IEEE Globecom '98 Internet Mini-Conference*, Sydney, Australia, November 1998.

[104] V. Padmanabhan and J. Mogul. Improving HTTP Latency. *Proc. of the Second International World Wide Web Conference*, Chicago, IL, USA, October 1994.

[105] V. N. Padmanabhan and J. C. Mogul. Using Predictive Prefetching to Improve World Wide Web Latency. *ACM SIGCOMM Computer Communication Review*, Vol. 26, No. 3, July 1996.

[106] V. N. Padmanabhan and L. Qiu. The Content and Access Dynamics of a Busy Web Site: Findings and Implications. In *Proc. of SIGCOMM '2000*, Stockholm, Sweden, August - September 2000.

[107] K. Park, G. Kim, and M. Crovella. On the Relationship between File Sizes, Transport Protocols and Self-Similar Network Traffic. *Proc. of ICNP '96*, Columbus, Ohio, USA, 1996.

[108] V. Paxson. Growth Trends in Wide Area TCP Connections. IEEE Network, Vol. 8, No. 4, pp. 8 - 17, July/August 1994.

[109] V. Paxson. Measurements and Analysis of End-to-End Internet Dynamics. PhD thesis, U.C. Berkeley, May 1996.

[110] L. Qiu, V. N. Padmanabhan, and G. Voelker. On the placement of Web server replicas. To appear in *IEEE INFOCOM'2001*, Anchorage, AK, USA, April 2001.

[111] M. Rabinovich. Issues in Web Content Replication Data Engineering Bulletin, Invited paper, Vol. 21, No. 4. December 1998.

[112] M. Rabinovich and A. Aggarwal. RaDaR: A scalable architecture for a global Web hosting service. *Proc. of the 8th International World Wide Web Conference*, Toronto, Canada, May 1999.

[113] M. Rabinovich, I. Rabinovich, R. Rajaraman, and A. Aggarwal. A dynamic object replication and migration protocol for an Internet hosting service. *Proc. of the IEEE Int. Conf. on Distributed Computing Systems*, May 1999.

[114] D. Rubenstein, J. Kurose, and D. Towsley. Detecting Shared Congestion of Flows Via End-to-end Measurement. *Proc. of SIGMETRICS '00*, Santa Clara, CA, USA, June 2000.

[115] S. Savage, N. Cardwell, and T. Anderson. The Case for Informed Transport Protocols. *Proc. of the 7th Workshop on Hot Topics in Operating Systems (HOTOS '99)*, Rio Rico, AZ, USA, March 1999.

[116] S. Seshan, M. Stemm, and R. Katz. SPAND: Shared Passive Network Performance Discovery. *Proc. of the 1st Usenix Symposium on Internet Technologies and Systems (USITS '97)*, Monterey, CA, USA, December 1997.

[117] T. Shepard and C. Partridge. When TCP Starts Up With Four Packets Into Only Three Buffers. RFC-2416, September 1998.

[118] D. Shmoys, E. Tardos, and K. Aardal. Approximation algorithms for facility location problems. In *Proc. of the 29th ACM Symposium on Theory of Computing*, pages 265 - 274, El Paso, TX, USA, May 1997.

[119] Squid Internet object cache. http://squid.nlanr.net.

[120] I. Stoica and H. Zhang. Providing Guaranteed Services Without Per Flow Management. *Proc. of SIGCOMM '99*, Boston, MA, USA, September 1999.

[121] M. B. Teitz and P. Bart. Heuristic Methods for Estimating the Generalized Vertex Median of a Weighted Graph. Operations Research, Vol. 16, 1968, pp. 995 - 961.

[122] R. Tewari, M. Dahlin, H. Vin, and J. Kay. Design considerations for distributed caching on the Internet. In *Proc. of the 19th IEEE. Int. Conf. on Distributed Computing Systems*, May 1999.

[123] K. Thompson, G. Miller, and R. Wilder. Wide-Area Internet Traffic Patterns and Characteristics. *IEEE Network*, 11(6):10-23, November 1997.

[124] J. Touch. TCP Control Block Interdependence. RFC-2140, April 1997.

[125] J. Touch. The LSAM Proxy Cache - a Multicast Distributed Virtual Cache.

[126] J. Touch. Web Caching and Replication - Research Issues. Internet daft, draft-ietf-wrec-res-00.text. http://www.wrec.org/Drafts/draft-ietf-wrec-res-00.txt.

[127] A. Vahdat, M. Dahlin, T. Anderson, and A. Aggarwal, Active Names: Flexible Location and Transport of Wide-Area Resources. In *Proc. of USITS '99*, Boulder, CO, USA, October 1999.

[128] V. Valloppillil and K. W. Ross. Cache array routing protocol v1.0. ftp://ftp.isi.edu/internet-drafts/draft-vinod-carp-v1-03.txt, Febuary 1998.

[129] V. Visweswaraiah and J. Heidemann, "Improving Restart of Idle TCP Connections," Technical Report 97-661, University of Southern California, November 1997.

[130] G. M. Voelker. Personal Communication, Feb 2000.

[131] W. Willinger, M. Taqqu, R. Sherman, and D. Wilson, "Self-Similarity through High Variability: Statistical Analysis of Ethernet LAN Traffic at the Source Level," *Proc. of SIGCOMM '95*, Cambridge, MA, USA, August - September 1995.

[132] P. Wolfe, and H. P. Crowder. Validation of Subgradient Optimization. Math Programming. Vol. 6, 1974, pp. 62 - 88.

[133] A. Wolman, G. M. Voelker, N. Sharma, N. Cardwell, M. Brown, T. Landray, D. Pinnel, A. Karlin, and H. Levy. Organization-based Analysis of Web-Object Sharing and Caching. In *Proc. of USITS '99*, Boulder, CO, USA, October 1999.

[134] A. Wolman, G. M. Voelker, N. Sharma, N. Cardwell, M. Brown, T. Landray, D. Pinnel, A. Karlin, and H. Levy. On the Scale and Performance of Cooperative Web Proxy Caching. In *Proc. of SOSP '99*, Charleston, SC, USA, December 1999.

[135] A. Wolman. Characterizing Web workload to improve performance. July 2000.

[136] H. Yu, L. Breslau, and S. Shenker. A Scalable Web Cache Consistency Architecture. In *Proc. of SIGCOMM '99*, Boston, MA, USA, September 1999.

[137] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee. How to Model an InternetWork. In *Proc. of INFOCOM '96*, San Francisco, CA, USA, March 1996.

[138] Y. Zhang, L. Qiu, and S. Keshav. Speeding Up Short Data Transfers: Theory, Architectural Support, and Simulation Results. In *Proc. of NOSSDAV '2000*, Chapel Hill, NC, USA, June 2000.