SCALABLE CONTENT DISTRIBUTION IN THE INTERNET

THÈSE NUMÉRO xxxx (2000)

PRÉSENTÉE AU DÉPARTEMENT DE SYSTÈMES DE COMMUNICATIONS

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Pablo Rodriguez Rodriguez

Ingénieur en Télécommunications,

Universidad Pública de Navarra (UPNA), Espagne

Composition du Jury:

Prof. A. Schiper, président du Jury

Prof. E.W. Biersack, directeur de thèse

Prof. J. Ararcil, corapporteur

Prof. J.Y. Le-Boudec, corapporteur

Prof. R. Molva, corapporteur

Dr. P. Nain, corapporteur

Lausanne, EPFL
October, 2000

Acknowledgments

First and foremost I would like to extend my deepest thanks and appreciation to my research advisor Professor Ernst W. Biersack. Ernst, is a person with enormous intuition, power of abstraction, and brilliant capabilities to identify good problems and easily come up with solutions. Moreover, he offers an extraordinary level of human warmth, encouragement, and most importantly faith and trust. During these years I learned a great amount from him, both on the professional as well as on the personal side. I hope to maintain a long lasting friendship with him as well as keep learning from his experience and advice in the years to come.

The work presented in this thesis is also the result of several fruitful and rewarding collaborations. I would like to thank Professor Keith W. Ross for his guidance and his valuable input in the analytical part, as well as in the first chapters of this thesis. I would also like to acknowledge the opportunity I was given to spend several months at AT&T Research Labs. I would like to specially thank Sandeep Sibal, my mentor at AT&T, for his great research input, excellent advice, and friendship. Similarly, I would like to thank other researchers at AT&T Labs – Fred Douglis, Misha Rabinovich, and Ramon Caceres – for their support. Also, great thanks to the different people who shared with me multiple thoughts, projects, and long discussions that influenced this work in multiple ways: Jörg Nonnenmacher, Christian Spanner, Andreas Kirpal, Guillermo Barrenechea, and Thomas Proell.

I should not forget the generous financial support that I received from the European Commission through these years in terms of a Marie Curie fellowship, and the great set of resources and opportunities that I found at Eurecom. Thanks to Didier Loisel, and David Trémouillac for their permanent cooperation and competence in any network or software administration issue. From Didier I learned that in the technical world “there exist no problems without solution: if you have not found the solution is because you have not looked carefully into it yet...”. Thanks to all the Btroup members and the other Eurecom colleagues for making these years very pleasant and for providing a friendly atmosphere. Also, thanks to Evelyne Biersack and to Guillaume Urvoy for having helped me with the French translations. Special acknowledgments to all the administrative staff who make an extraordinary effort to ensure that everything at Eurecom works smoothly and properly.

My special appreciation and thanks go to Despina, to whom I am indebted for numerous reasons, including her Athenian advice, encouragement, and support on innumerable occasions.

Foremost, I would like to thank Karel for her patience, love, and care during these years. I would also like to express my gratitude to her family for their continuous encouragement and warmth, especially to her brother, from whom I learned how to fight for important things in life with courage and faith.

Finally, and most importantly it is to my brother and my parents that I owe the satisfaction and joy of writing these lines. Their ever present support, love, innumerable sacrifices, and endless encouragement made it possible to arrive to this point. It is to them that I have the pleasure to dedicate this thesis.

Scalable Content Distribution in the Internet

Pablo Rodriguez Rodriguez

Abstract

The explosive growth of the Internet is a major driver of the telecommunications market today. In future years, the number of online users will exhibit strong growth in every geographic region. In addition to a larger number of Internet users, the amount of content as well as the richness of the content are rapidly increasing. Users around the world are also beginning to have broadband access services. Higher access rates to the Internet allow users to view more and richer content and exacerbate the congestion problems in the network. There is, therefore, a strong need for scalable and efficient content distribution schemes.

This thesis makes several contributions to the area of content distribution. We provide theoretical formulation and analytical models to evaluate and study content distribution systems. The models allow the evaluation of important metrics of content distribution systems such as hit rate, disk space, client latency, server load, and bandwidth usage, under scenarios including caching, multicasting, and satellite broadcasting.

We compare and study different schemes to scale the distribution of content to a large number of clients. In particular we consider caching and multicast as two natural schemes for scalable content distribution. We derive simple analytical models to study the performance of caching and multicast in terms of latency and bandwidth and determine when caching outperforms multicast. Further, we extend our analysis of caching and multicast to study the impact of a caching hierarchy in reducing the traffic required to keep content up-to-date, and present how to combine caching and multicast. Based on this study we introduce an extremely lightweight solution that allows the ad-hoc and automatic deployment of overlay networks of caches.

Satellite broadcasting is another natural scheme for scalable content distribution. We discuss a novel approach that combines a satellite distribution with caches to bring popular content to the network edge. We use analytical models and trace-driven simulations to

calculate upper bounds on the performance of a cache-satellite distribution, considering the impact of many different parameters over a large range of values.

We then consider the problem of strong consistency, that is, how to provide clients with the most up-to-date content. We present a new technique that automatically decides on the best mechanism for strong consistency based on the request rate and modification rate of the content. We evaluate this new technique in terms of bandwidth and latency and compare its performance with previous approaches.

When popular content is replicated in multiple network locations clients should be transparently redirected to the optimum replica. We develop a new solution where clients connect in parallel to multiple replica sites to provide fast access to replicated content and alleviate the problem of replica selection. We study the performance of such a parallel-access scheme and present the results of a real implementation.

Scalable Content Distribution in the Internet

Pablo Rodriguez Rodriguez

Résumé

La croissance massive de l'Internet est le fait marquant sur le marché des télécommunications aujourd'hui. Le nombre d'utilisateurs au niveau mondial va fortement augmenter, avec pour conséquence des contraintes de plus en plus importantes sur l'architecture existante. À la croissance du nombre de clients s'ajoute celle des contenus en terme de quantité mais également de diversité. De plus en plus d'utilisateurs dans le monde commencent également à avoir un accès haut débit, ce qu'est évidemment le premier pas vers le multimédia à domicile, mais a aussi pour conséquence d'aggraver les problèmes de congestion. Il y a donc actuellement une forte demande pour le développement de services de distribution de contenus efficaces et scalables.

Cette thèse apporte plusieurs contributions dans le domaine de la distribution de contenus sur l'Internet. Nous commençons par fournir une formulation théorique du problème et développer un modèle analytique qui nous permette d'étudier de tels services. Ce dernier permet l'évaluation de nombreux paramètres clés des systèmes à base de cache - tels que le taux de réussite, l'espace disque nécessaire, le temps de latence, la charge du serveur ou la bande passante nécessaire - dans différents contextes - multi-points et diffusion satellite.

Nous avons étudié et comparé différents systèmes visant à assurer une diffusion scalable des contenus. En particulier les systèmes de caches et de transmission multi-points, qui semblent deux moyens naturels d'assurer une diffusion scalable de contenus. Nous avons évalué de manière analytique les performances de ces solutions en terme de temps de latence et de débit requis, ce qui nous a permis de déterminer la durée minimale de validité des documents au delà de laquelle un système de cache devient plus efficace qu'un système de transmission multi-points. Nous avons étendu cette comparaison multi-points/caches au cas des caches hiérarchiques en étudiant la capacité de ces derniers à réduire le surcoût dû au maintien à jour des documents, et la manière de combiner les deux approches (caches et multi-points).

Nous présentons enfin une solution extrêmement souple permettant un déploiement ad-hoc et automatique d'un système de cache pour des réseaux de type overlay.

La diffusion satellite est une autre solution naturelle pour assurer une diffusion scalable de contenus. Nous présentons une approche nouvelle qui combine distribution satellite et caches pour amener les documents les plus populaires au plus près des clients. Nous employons un modèle analytique et des simulations basées sur des traces réelles pour calculer des bornes supérieures sur les performances de la diffusion satellite. Nous analysons également l'impact de nombreux paramètres.

Nous étudions ensuite le problème de la fourniture de la version la plus actuelle d'un document à un client: nous avons ainsi développé une nouvelle approche qui permet de décider de manière automatique du meilleur moyen à mettre en œuvre pour assurer le maintien à jour en fonction du taux d'accès et du temps de vie du document. Apres avoir évalué cette nouvelle technique en terme de bande passante et de temps de latence, nous la comparons aux approches précédemment envisagées. Quand les contenus les plus populaires sont répliqués dans de multiples serveurs, les clients doivent être dirigés de manière transparente vers le meilleur serveur. Nous développons une nouvelle solution où les clients se connectent en parallèle aux multiples serveurs pour afin d'avoir un accès rapide aux contenus répliqués et de contourner le problème de la sélection du serveur. Nous étudions la performance d'un tel système d'accès en parallèle et présentons les résultats d'une implémentation réelle.

Contents

List of Figures

List of Tables

List of Variables

λ	Request rate for a single document
β	Request rate for multiple documents
N	Number of documents
t	Update period of a document
Δ	Mean update period of a document
$\lambda\Delta$	Mean number of requests per update period
H	Height of the network tree modeling an Internet Service Provider (ISP)
O	Outdegree of the network tree
L	Specific level of the network tree
z	Number of network hops of the International Path
C	Link Capacity
μ	Document transmission rate
T	Latency to receive a document
S	Document size
ϕ	Age of a document
σ	Max-age of a document
M	Number of mirror sites
B	Number of document blocks
RTT	Network round-trip time
α	Skew parameter of the Zipf distribution

Chapter 1

Introduction

1.1 The Need for Scalable Content Distribution

Todays Internet with millions of clients accessing a large amount of content suffers frequently from high response times due to overload of popular servers or congestion of network links. The Web offers a simple way to access pre-stored content such as Web pages or audio and video files, and rapidly became the most popular application of the Internet. In the near future, broadcasting life events or streaming video files is expected to be the new killer application of the Internet. The explosive growth of the Web has not always been followed by the corresponding upgrades in network and server resources and has created several *Internet bottlenecks*. The main Internet bottlenecks are the lack of sufficient bandwidth at the "last mile" (i.e. the bandwidth that an end-user utilizes to connect to an Internet Service Provider (ISP), the congestion inside the ISPs, the congestion in the network access points (NAPs) or peering points that interconnect ISPs, and the high load on the content provider's servers.

Until recently, much of the Internet's ”slowness” was attributed to the last mile. However, many home-based Internet users are finally gaining access to broadband Internet connections, such as xDSL, cable, and to a lesser extent, fixed wireless and satellite. Thus, the bottleneck is moving from the last mile to other areas of the infrastructure. Clients with broadband connections exacerbate the problems of congestion in other areas of the Internet, as they tend to increase the traffic load and demand richer content. At the ISP and content provider level, the common remedy to solve bottlenecks has traditionally been adding bandwidth and/or servers. However, simply increasing the capacity of the network links, or adding more servers at the content provider sites in an arbitrary fashion, is only a short term solution since there can always appear new applications that will demand more resources. Besides, adding bandwidth or servers is very often neither efficient nor cost effective since system resources may be poorly utilized. A more serious problem happens at the access points between ISPs. Since most Internet traffic is exchanged between networks for free, ISPs have an incentive to improve intra-ISP network performance, but not inter-ISP network performance. Concerning intra-ISP traffic, ISPs have an incentive to minimize intra-ISP resource utilization to accommodate more customers. ISPs frequently redirect traffic destined to another ISP to the closest peering point. These redirection decisions are made in a rather arbitrary fashion, causing poor performance. Thus, even if ISPs make an extraordinary effort to improve their intra-network communication, it may have minimal impact on the quality perceived by the client whose communications need to traverse several ISPs.

Solving the Internet bottlenecks requires a combination of solutions. Fixing the last-mile problem or increasing the resources in the ISPs is only a partial solution. There is a strong need for intelligent and *scalable content distribution* solutions that improve the Internet performance in a cost-effective and efficient manner. We study and propose several new schemes for providing scalable content distribution in the Internet. We consider the whole spectrum of solutions putting them in a global perspective and discussing the tradeoffs between the different approaches and how they can be combined. The solutions considered include caching, replication, load balancing, multicast, and satellite distribution.

1.2 Issues in Content Distribution

There are several challenges and issues involved to provide an efficient content distribution architecture. Next, we outline some of the most important issues.

- Since there is a *large amount of content* that should be potentially delivered to *many clients*, it is necessary to provide a **scalable** infrastructure. Scalability is defined as the independence of the performance of the system from the client population size and the amount of content.

- Since popular content usually changes on a regular basis, it is very important to provide clients with the most up-to-date version of that content. Thus, when content changes at the content provider, copies outdated should be automatically and intelligently updated with a copy of the new content. Providing clients always with the most up-to-date information is known as **strong consistency**.

- Various copies of the same content can be placed at different network locations. Thus, clients should be dynamically re-directed to the ”best” copy, that is, the copy that would offer the fastest downloading time. Efficient **client redirection** mechanisms are very important for any scalable content distribution solution.

The research conducted in this thesis focuses on the study, design, and evaluation of different scalable content distribution approaches, paying special attention to the issues described above.

1.3 Approaches to Content Distribution

One of the first approaches to content distribution was the introduction of *client caches* either implemented as host-local disks or as in-memory caches. Although client caches can achieve reasonable *hit rates*, i.e. the percentage of requests satisfied from the cache, soon it was clear that caches shared by many people with similar interests would yield higher hit rates. The first widely available software providing this functionality was the CERN proxy [60], created in early 1994 at the birthplace of the World Wide Web, the CERN laboratories in Geneva. This software gained widespread acceptance and served as a reference implementation to many caching proxies[1] developed later on. However, the CERN cache failed to provide cooperative caching schemes. The CERN cache could only be configured to use one parent cache. Moreover, it could not gracefully deal with the failure of such parent caches. HENSA [84], was one of the first and best documented approaches to building a coherent large caching infrastructure. The HENSA cache was started in late 1993 in the University of Kent. The goal was to provide an efficient national cache infrastructure for the UK. One single, centrally administered cache was created to handle over one million cache requests per day. The approach of a centrally administered caching service subsequently faced several barriers inherent to a non-distributed approach. Harvest developed in 1995 a much more sophisticated caching system, resulted from the Harvest information discovery project [18]. The Harvest approach sharply contrasts in concept from the HENSA approach by enabling administrators to span a tree of cooperating caches over wide area distances. The technical improvements implemented in the Harvest cache aim at optimizing nearly any component on the critical path from user request to delivery of the data. The public descendant of the Harvest cache, Squid [88], soon became the most popular publicly available caching software. Additionally, the Harvest cache took caching one step further towards the original server by providing a mode in which the cache could be run as an *accelerator* for origin servers.

During the last years there has been a lot of research to improve the performance of Web caches and provide new and more efficient caching sharing schemes [37] [80] [64] [72] [86]. However, traditional Web caching was created with a very strong ISP oriented point of view. Web caches were installed by ISPs to save bandwidth and delay the process of upgrading their access links to other ISPs. In the USA, bandwidth was very cheap and ISPs had little incentive to deploy caches. In Europe, on the other hand, bandwidth was much more expensive and many ISPs rapidly started deploying caches. However, Web caches administered by ISPs use their own time-to-live heuristics for the documents stored in the cache, and may provide clients with stale content. Popular content providers were very concerned with the fact that their content was delivered to the clients without them having control over

[1]In the rest of the thesis we use the term proxy, cache, and proxy cache interchangeably, since the proxying and caching functions are co-located in a single entity.

the freshness of the content. This situation made the European Commission consider to ban Web caching in early 1999. In response to the ISP's control of the cache, content providers often misuse or abuse features of the HTTP protocol [41] to make content uncacheable.

Since many ISPs, specially in the USA, were not deploying caches and popular content providers were defeating caching where deployed, popular sites started experiencing high loads and flash crowds. Thus, some large content providers decided to deploy their own network of *mirror servers/reverse proxy caches* that delivered only their content. Since mirror servers are directly administered by content providers, content providers can fully control what is being delivered to the clients. Given a network of mirror servers, clients should be automatically and transparently redirected to the optimal server. The most popular techniques to redirect clients to a server are application-level redirection using the HTTP protocol and DNS redirection. Combining efficient redirection techniques with a network of mirror servers provided a very efficient solution for many content providers (e.g., CNN, Microsoft, WWW Consortium). However, building and running such an infrastructure requires a considerable amount of effort at the content provider's site. Content providers would rather out-source the operation of the content distribution infrastructure and focus on content production.

Recently several companies started providing content distribution infrastructures as a service for content providers. Content Distribution Networks (CDNs) such as Akamai [2] started deploying private overlay networks of caches around the world to circumvent much of the public Internet and provide fast and reliable access for content provider's clients. Such a content distribution network can be shared by multiple content providers, thus, reducing the costs of running such network. Akamai Technologies, founded by two MIT scientists in August 1998, rapidly established itself as the market leader in content distribution. Akamai has deployed a broad global network for static content and streaming media, with several hundreds of nodes that cover over 40 countries. Since the creation of Akamai, the content distribution market has experienced a tremendous growth and is expected to continue during the coming years, integrating new technologies such as multicast or satellite broadcast.

1.4 Research Scope

There are multiple and interesting problems related to content distribution, which span many areas. In this thesis we consider fundamental content distribution problems that require a careful study and analysis to understand the benefits and trade-offs of the different solutions. The research performed pays special attention to scalability issues. In the following, we describe in more detail some of the research problems considered.

A natural way to scale the distribution of content to a large number of clients is using **IP-multicast**. With IP-multicast [36] [56] the same content traverses each link on the path from the server to the clients only once, thus, sharing all common paths and saving bandwidth. However, IP-multicast requires fundamental changes in the network and poses still some technical issues that need to be resolved, such as congestion control [31]. Thus, a global IP-multicast infrastructure in the Internet maybe still years away. Meanwhile, the price of disks is decreasing at a faster rate than network bandwidth [12] and it is cheaper to store a document locally than to retrieve it twice from the origin server. Also, storing a document locally drastically decreases the downloading times and reduces the load in the servers and the network. Therefore, **Web caching** has rapidly become an important component of a scalable content distribution.

With Web caching, each document request is first sent to the cache; if the document is present in the cache, the document is delivered to the client; otherwise, the cache retrieves the document from the origin server, places a copy in its storage, and forwards the document to the client. With the deployment of content distribution networks (CDNs) [2], caching was turned into a service for content providers. The premise of CDNs is that it is cheaper and faster to serve data-rich content from servers located at the ”edge” of the network, than from servers in the middle of the network. The business model of CDNs is targeted to content providers and not to ISPs, since content providers have a big incentive to improve user’s browsing experience using caches at the edge of the network. Though Web caching for CDNs and Web caching deployed by ISPs rely on two different business models, most of the technical issues and challenges remain the same.

We first consider IP multicast and application-level multicast using hierarchical caching as two approaches to scalable content distribution to either multiple caches or to the clients directly. With **hierarchical caching** [23] caches are placed at different network levels (e.g., institutional, regional, national). At the bottom level of the hierarchy there are client caches. When a request is not satisfied by a client cache, the request is redirected to the institutional cache. If the document is not present at the institutional level, the request travels to the regional cache which in turn forwards unsatisfied requests to the national cache. If the document is not present at any cache level, the national cache contacts the origin server. When the document is found, either at a cache or at the origin server, it travels down the hierarchy, leaving a copy at each of the intermediate caches. Further requests for the same document travel up the caching hierarchy until the request finds the document. Using analytical models we determine when to use IP multicast or hierarchical caching to minimize the latency experienced by the clients or the bandwidth usage inside an ISP.

To scale a content distribution scheme, we also consider a **satellite distribution** of Web documents combined with a caching infrastructure. Web caches store content locally to satisfy multiple requests for the same document from the same client (i.e., temporal locality) or from different clients (i.e., geographical locality). However, since the number of clients connected to a single cache can be rather small and the amount of information available in the Web is rapidly increasing, the locality of client requests can be very poor and the performance of the cache quite modest. The hit rate, i.e. the percentage of requests that can be served from previously cached document copies, at an institutional cache typically ranges between 30% and 50% [79]. The hit rate of a Web cache can be increased significantly by sharing the interests of a larger community [23]; the more people are accessing the same cache, the higher the probability that a given document is present in the cache. Using a cache-satellite distribution, a cache ends up storing the documents requested by a much larger community, approximately the total number of clients in all caches connected to the satellite content distribution service. We investigate the feasibility and requirements of a cache-satellite scheme in terms of disk space at the caches, bandwidth required at the satellite, and hit rate at the caches. Using analytical models and trace-driven simulations we study the performance of a cache-satellite distribution and compare it to the performance of a caching hierarchy.

Since popular content changes on a regular basis, and people have begun to rely on the Internet for conducting business and other critical operations, it is imperative to provide clients with the most up-to-date information (i.e. **strong consistency**). Traditional Web caches have been ineffective in this area, since caches administered by ISPs may provide clients with stale information. With content distribution networks, on the other hand, content providers can ensure that the content delivered to the clients from the caches is always fresh, however, the mechanisms to provide strong consistency are various. The primary mechanisms for strong consistency are: client validation, server invalidation and replication. Instead of focusing on a single mechanism to provide strong consistency, we considered the whole range of possibilities and tried to understand when to use each of them. We present a dynamic approach that automatically decides on the best control policy to keep content up-to-date based on the popularity and rate of change of the content.

The large deployment of CDNs and caches will cause the same information to be replicated in multiple network locations. However, choosing the optimal replica is a non-trivial task. Choosing the best replica has been the subject of research during the last years. Techniques like dynamically probing, or statistical recording show that the choice of the best replica is not always obvious and that performance can dramatically vary depending on the replica selected. We take a different approach for selecting the best replica: instead of using a selection algorithm to pick one replica site, we propose a scheme in which clients access multiple replica sites in parallel to download the content. We study the performance of such parallel-access scheme and present the results of a real implementation.

1.5 Issues not Covered in this Thesis

There are several important issues related to content distribution and Web caching in general that we do not explore in this thesis.

- *Cache performance*: Tuning the cache performance is a critical task since caches need to handle many requests per second and provide a similar performance than dedicated Web servers. The main desired characteristics of a cache are high throughput, small response times, and high hit rates. Several caching bakeoffs [6] hosted by the National Laboratory for Applied Network Research (NLANR [6]) have recently tested independent implementations of different caches. The results of this bakeoffs show that there already exist high performance caches at a reasonable cost.

- *Cache Cooperation Schemes*: To increase the number of documents delivered from a caches, multiple caches can cooperate. Cache cooperation has been subject of research for several years and there are already very efficient cache-sharing protocols. However, the deployment of these cache-sharing schemes in a wide area environment has not been very significant. Caches that already have a large client population do not experience significant improvements by cooperating with other caches since the locality of reference among clients is already very high.

- *Replacement policies*: An important issue of a cache is deciding which documents to evict once the cache is full. Several sophisticated replacement policies have been proposed that take into account factors such as the popularity of a document, the document size, and the locality of reference between client requests for the same document [19] [17] [68]. However, given the current storage capacity of disks (e.g., several TBytes), and the fact that few documents account for most of the requests, it is safe to assume that caches can have virtually an infinite storage capacity. This relaxes the need for efficient cache replacement policies. Nevertheless, cache replacement policies may become very relevant for audio and video content, where the average size of a document is orders of magnitude larger than for regular Web pages.

- *Algorithms for intelligent prefetching*: Predicting future client requests can be used to perform intelligent content distribution mechanisms and improve client browsing experience. One simple prefetching mechanism is to preload those in-lined pages referenced in a given page. Other more sophisticated prefetching mechanisms can use statistics about client access patterns to intelligently decide which documents to prefetch. There is a clear compromise between the bandwidth used to prefetch documents and the improved client's latency; clients can experience better latencies at the expenses of prefetching more documents that might be never requested, thus, wasting bandwidth. Bandwidth usage is an important issue for terrestrial networks, but it might be less important for satellite networks where bandwidth is plentiful.

- *Security Issues*: An important issue of a content distribution architecture is to provide strong security features. Information should be protected to avoid that content for which clients are required to pay is not freely replicated and distributed. Another important issue is ensuring that a content distribution service is always available. Content distribution networks can provide solutions to avoid *denial of service attacks* by implementing request queuing mechanisms at the caches.

1.6 Thesis Contributions

The thesis makes several contributions. It first develops a **theoretical formulation and analytical models** to evaluate and study content distribution systems. The models allow the evaluation of important metrics of a caching system such as hit rate, disk space, client latency, server load, and bandwidth usage, under scenarios including multicasting and satellite broadcasting. The analytical results are validated using real traces to determine the accuracy of our model. The analytical results can be used by Internet Service Providers as well as Content Distribution Networks for an initial specification of the requirements of their content distribution infrastructure.

The second contribution is the **comparison** of two natural schemes to scale a content distribution system: **caching** and **multicast**. We develop analytical models to study the performance of both schemes in terms of latency and bandwidth. Our analytical results show that hierarchical caching outperforms reliable multicast to distribute most of the content in the Internet (i.e. all content that changes once every few minutes of less). Based on these results, we further investigate the advantages of a caching hierarchy when caches poll the origin servers on each client request to provide always fresh content. Using analytical models and trace-driven simulations we show that a caching hierarchy can effectively reduce the traffic in the network as well as the number of polls to the origin servers as soon as origin servers set a small time-to-live during which the content does not change. We also developed new schemes that combine hierarchical caching and multicast to push a large amount content to many clients.

The third contribution is the study of a **satellite distribution** combined with a Web caching system. We use analytical models to calculate upper bounds on the performance of a cache-satellite distribution and study the impact of different parameters over a large range of values. In particular we derive expressions to calculate the maximum achievable hit rate in a cache connected to a cache-satellite distribution, the latency experienced by clients, the disk space required at the caches, and the bandwidth needed at the satellite. We show that a cache-satellite distribution is a very efficient content distribution solution and can significantly improve the performance for content providers, ISPs, and clients.

The fourth contribution is the design of **SPREAD** – a new architecture for distributing and maintaining up-to-date Web content that is largely based on our previous results on caching hierarchies/infrastructures. SPREAD uses a new class of proxies (*Translucent Proxies TPOT*) that automatically self-configure themselves to form scalable application-level content distribution architectures. SPREAD's overlay network of proxies can avoid most of the problems that plague IP-multicast, such as congestion control and inter-domain routing. Another important facet of SPREAD is its approach to assuring strong consistency in order to provide clients with the most up-to-date content. SPREAD uses primarily three mechanisms for providing strong consistency: client validation, server invalidation, and replication. Instead of arbitrarily selecting a consistency mechanism, each proxy within SPREAD autonomously decides on the best mechanism to provide strong consistency based on the popularity and modification rate of the content. We show that SPREAD results in significantly higher performance in terms of bandwidth and latency than any other consistency mechanism.

Finally, the fifth contribution is the design of a **parallel-access** scheme to replicated information as an alternative to server selection algorithms. With a parallel-access, clients connect to the replica sites using unicast TCP connections and dynamically request different pieces of a single document from different sites. We implement and test the performance of a parallel-access under real conditions. The experimental results show that other than avoiding the server selection algorithms, our scheme dramatically improves the perceived client latency, increases resilience against server failure and network congestion, and automatically balances the load between the different sites.

1.7 Thesis Organization

The remaining of the thesis is composed of five research chapters and a final concluding chapter. The following chapters are mostly self-contained. In Chapter 2 we study and compare the performance of hierarchical caching versus multicast. In this chapter we develop analytical models that will be used in multiple parts in the thesis. In Chapter 3 we further extend the analysis of hierarchical caching and present different advantages of hierarchical caching to push content to a large number of clients. In Chapter 4 we study the performance and feasibility of a satellite distribution combined with a caching infrastructure. We derive analytical models to study a cache-satellite distribution and validate our models with trace-driven simulations. Chapter 5 presents SPREAD, a new architecture for distributing and maintaining up-to-date content. We describe and analyze how SPREAD dynamically builds hierarchical trees for scalable content distribution and its approach for providing strong consistency. Chapter 6 presents a parallel-access scheme for popular content that is replicated in multiple network locations. We present the design of a parallel-access scheme and the results of a real implementation. Finally, Chapter 7 summarizes the results and contributions of this thesis and gives an outlook.

Chapter 2

Improving the WWW: Caching or Multicast?

In this chapter we consider two schemes to scale the distribution of popular Web documents. In the first scheme, the sender repeatedly multicasts a Web document and receivers asynchronously join the corresponding multicast tree to receive a copy. In the second scheme, the document is distributed to the receivers through a hierarchy of Web caches. We develop analytical models for both schemes, and use the models to compare the two schemes in terms of latency and bandwidth usage. We find that except for documents that change very frequently, hierarchical caching gives lower latency, and uses less bandwidth than multicast. Only for rapidly changing documents, multicast outperforms hierarchical caching.

2.1 Introduction

The response time to retrieve a Web document can be very high, particularly when a document is retrieved over a transoceanic link. For example, from Europe it can take minutes to retrieve a small document from North America during local working hours. Even within a country latency can be unsatisfactory when trying to retrieve a document from a congested server during busy hours.

It is therefore of great interest to implement schemes in the WWW that reduce latency. One popular scheme is to use shared caches within a network (in addition to the local Web cache at the client [63] [91]). A shared Web cache can be placed at the institutional, regional, or even national level. To illustrate the idea, consider a large institution, such as a large university, confined to a contiguous geographical area. The clients of such an institution are typically interconnected with a low-cost high-speed LAN. Suppose the institution also attaches a shared Web cache to the LAN. In this common scenario, each request is first sent to the institution's cache; if the document is present in the cache, the document is delivered to the client over the high-speed LAN; otherwise, the cache retrieves the document from the origin server, places a copy in its storage, and forwards the document to the client. If the hit rate at the institutional cache is high, then the institution enjoys significant reductions in average latency and bandwidth usage. However, if a document – such as a news report, a weather forecast, or a stock-market quote – is updated frequently, then caching of the document may be of little benefit, and may in fact actually increase latency.

An alternative way of dealing with popular and frequently-changing documents is to distribute them with **continuous multicast push (CMP)** [76] [9]. Here, a server housing a popular and frequently-changing document continuously multicasts the latest version of the document on a multicast address. Clients tune into the multicast group for the time required to receive the document and then leave the multicast group. The Web server should only consider using CMP for highly-popular and frequently changing documents. (Our analysis shall show that the less frequently changing documents should be distributed with hierarchical caching.) A CMP distribution does not suffer from problems of over-loaded servers or caches. The multicast server does not deal directly with the receivers, reducing the server complexity and thus scaling very well. Receivers always receive the last available update of the document without polling the origin server on each request. Additionally, a multicast distribution uses bandwidth efficiently by sharing *all* common paths between the source and the receivers. Thus, CMP is an attractive scheme to deliver hot-changing documents.

An obvious disadvantage is that CMP requires a that the network connecting a server with its clients is multicast capable: a single packet sent by a server will be forwarded along the multicast tree (see Figure 2.1).

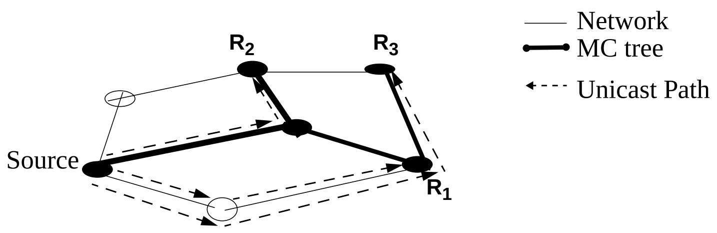


Figure 2.1: Network topology with multicast tree.

Where the multicast tree forks off, the multicast router will replicate the packets and send a copy on every outgoing branch of the multicast tree. Multicast routers on the Internet were first introduced via an overlay network called *MBONE* [36] [56] consisting of computers that executed the multicast routing software and that were connected via tunnels. While today the multicast routing software is part of any new router that is installed, not all the existing routers have been enabled to be multicast capable. Therefore, multicast routing on the Internet is not yet everywhere available [44].

Yet another means to distribute Web documents is to use **hierarchical caching**. In this scheme, shared caches are present at the institutional, regional, and national levels. Each client points its browser to its institutional cache, each institutional cache points to its regional cache, and each regional cache points to its national cache. Thus, when a client desires a document, a series of requests is sent up the caching hierarchy until an up-to-date copy of the document is found. When the document is found, either at a cache or at the origin server, it travels down the hierarchy, leaving a copy at each of the intermediate caches. (Again, multiple caches can be placed within each ISP at any tier to improve performance and increase storage capacity.) It is interesting to note that *hierarchical caching mimics multicast*. For example, to distribute a popular document from a Web server to M clients, it is not necessary for the server to send M copies of the document. Instead the server sends at most one copy to each of the national caches; each national cache sends at most one copy to each of the regional caches that point to the national cache, etc. (This sending is done asynchronously, driven by the client request pattern.) Thus, the caches act as *application-layer multicast* nodes, and the documents are sent from cache to cache by tunneling across routers.

Of course, caching requires cache servers to be purchased – perhaps a large number for each ISP in order to have sufficient storage, processing power, and access bandwidth. In fact, the effort of installing a caching hierarchy resembles the effort that was required to put in place the first experimental multicast overlay network, MBONE. But ISPs and Content Distribution Networks are prepared to pay the cost, as they are currently aggressively deploying caches [12] [2]. Hierarchical caching can be deployed much more easily than IP multicast since it operates at the application layer rather than at the network layer.

Next we compare the distribution of *hot and changing documents* by CMP and by hierarchical caching. We develop analytical models for both CMP and for hierarchical caching. We suppose that N_{HC} hot-changing documents have been identified. To simplify the analysis and to not obscure the key points, we assume that each of N_{HC} documents is updated at the same rate, e.g., once every minute. For the CMP model, we distribute these N_{HC} hot-changing documents with CMP; the less frequently-changing documents are distributed within an existing caching hierarchy. For the caching-hierarchy model, we assume all cacheable documents are distributed through the cache hierarchy, including the N_{HC} hot-changing documents. In this chapter we pay a particular attention to the impact of the queuing delays experienced on the caches due to their limited access bandwidth to the Internet. After developing performance models for these two distribution schemes, we attempt to shed some insight on when it is preferable to CMP the N_{HC} hot-changing documents or preferable to simply distribute all the documents through a caching hierarchy.

This chapter is organized as follows. In Section 2.2 we describe our specific model for comparing CMP to hierarchical caching. In Sections 2.3 and 2.4 we provide a latency analysis for CMP and hierarchical caching. In Section 2.5 we present a numerical comparison of the two schemes. In Section 2.6 we provide a bandwidth utilization analysis for the two schemes. In Section 2.7 we summarize our findings and briefly discuss how integrating caching with multicasting can improve performance.

2.2 The Model

As shown in Figure 2.2, the Internet connecting the server and the receivers can be modeled as a hierarchy of ISPs, each ISP with its own autonomous administration. We shall make the reasonable assumption that the Internet hierarchy consists of three tiers of ISPs: institutional networks, regional networks, and national backbones. All of the clients are connected to the institutional networks (typically via LAN or modem connection); the institutional networks are connected to the regional networks; the regional networks are connected to the national networks. The national networks are also connected, sometimes by transoceanic links. We shall focus on a model with two national networks, with one of the national networks containing all of the clients and the other national network containing the origin servers.

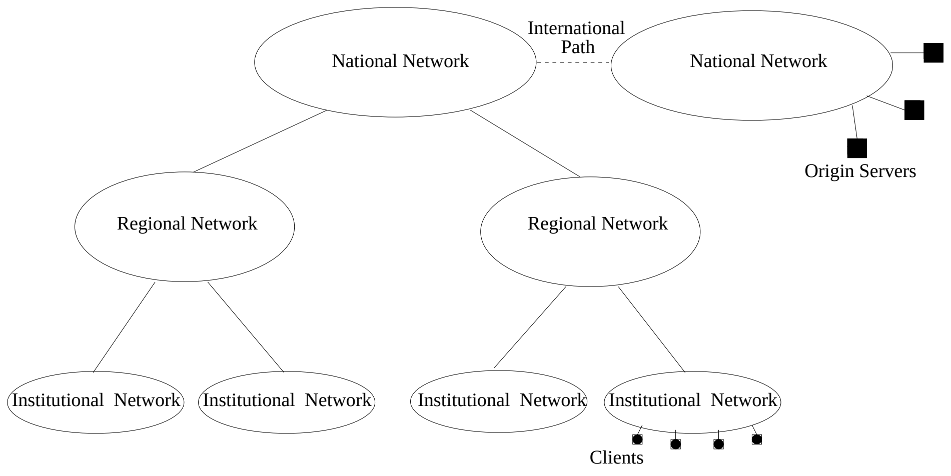


Figure 2.2: Network topology

In order to have a common basis for the comparison of caching versus multicast, as shown in Figure 2.3 we model the underlying network topology of the national and regional networks as a full O-ary tree (a full O-ary tree has proved to be a good model for network topologies, providing very realistic results [69]). Let O be the nodal outdegree of the tree. Let H be the number of network links between the root node of a national network and the root node of a regional network. H is also the number of links between the root node of a regional network and the root node of an institutional network. Let z be the number of links

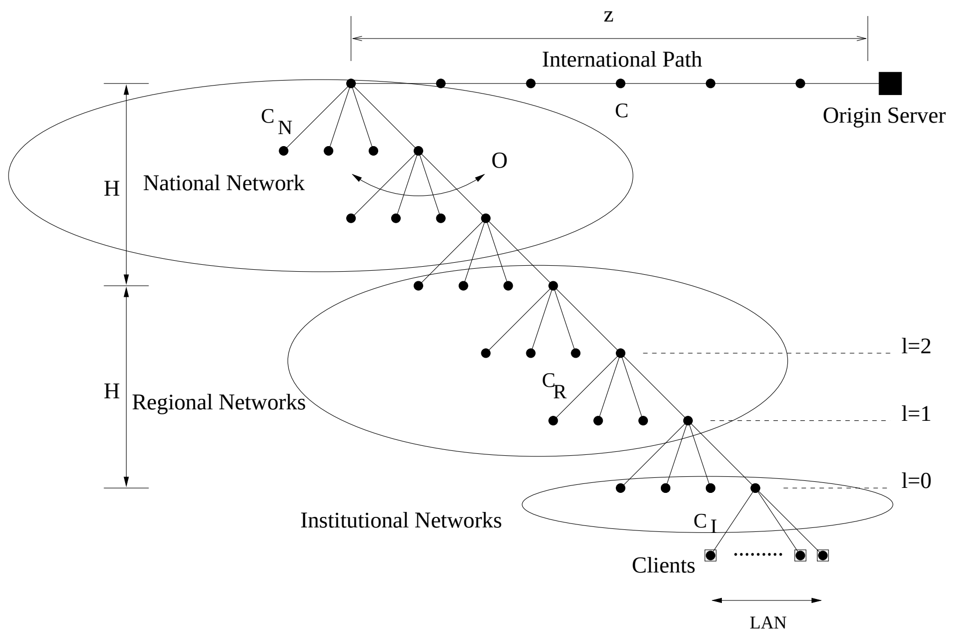


Figure 2.3: The tree model.

between a origin server and root node (i.e., the international path). Let d be the propagation delay on one link, homogeneous for all links. Let l be the level of the tree $0 \leq l \leq 2H + z$, where $l = 0$ represents the top node of the institutional network, and $l = 2H + z$ represents the origin server.

We assume that bandwidth is homogeneous within each ISP. Let C_I, C_R, and C_N be the **bandwidth capacity** of the links at the institutional, regional, and national networks. Let C be the bottleneck link capacity on the international path. Receivers are only on the leaves of the tree and not on the intermediate nodes. We assume that the network is a *lossless* network.

2.2.1 Document Model

In order to not obscure the key points, we make a number of simplifying assumptions. We assume that all documents are of the same size, S bytes (we consider a document to be a Web page or an in-lined image). We assume that each institution issues requests at a rate of β_{LAN}. We assume that there are N_{HC} hot-changing documents, all of which being candidates for CMP. From each institution the request rate for any one of the hot-changing documents is the same and is denoted by λ_I. We assume the λ_I is Poisson distributed. The assumption of Poisson arrivals is a reasonable one [45], [10] [68]. The total request rate from an institution for the hot-changing documents is $\beta_{LAN}^{HC} = N_{HC}\lambda_I$. The rate of the remaining *"background traffic"* is $\beta_{LAN}^{B} = \beta_{LAN} - \beta_{LAN}^{HC}$. Finally, let Δ be the update period of a hot-changing document. Initially we assume that all hot-changing documents change periodically every Δ seconds. In this case, caches do not need to contact the origin server to check for the document's consistency and the N_{HC} hot-changing documents can be removed from the caches every Δ seconds. We shall also consider non-periodic updates. For notational convenience, we use

$$\lambda_l = \lambda_I \quad for\ l = 0$$
$$\lambda_l = \lambda_I \cdot O^{l-1} \quad for\ l \geq 1,$$

for the aggregate request rate from all institutions below the multicast tree rooted at level l. Also $\lambda_{tot} = \lambda_I O^{2H}$ denotes the total request rate, aggregated over all institutions.

2.2.2 Hierarchical Caching Model

Caches are usually placed at the access points between two different networks to reduce the cost of traveling through a new network. As shown in Figure 2.4, we make this assumption for all of the network levels. In one country there is one national network with one (logical) national cache. There are O^H regional networks and every one has one (logical) regional cache. There are O^{2H} institutional networks and every one has one (logical) institutional cache. Caches are placed on height 0 of the tree (level 1 in the cache hierarchy), height H of the tree (level 2 in the cache hierarchy), and height $2H$ of the tree (level 3 of the hierarchy). If a requested document is not found in the cache hierarchy the national cache requests the document directly from the server.

Caches are connected to their ISPs via access links. We assume that the capacity of the access link at every level is equal to the network link capacity at that level, i.e., C_N, C_R, and C_I for the respective levels.

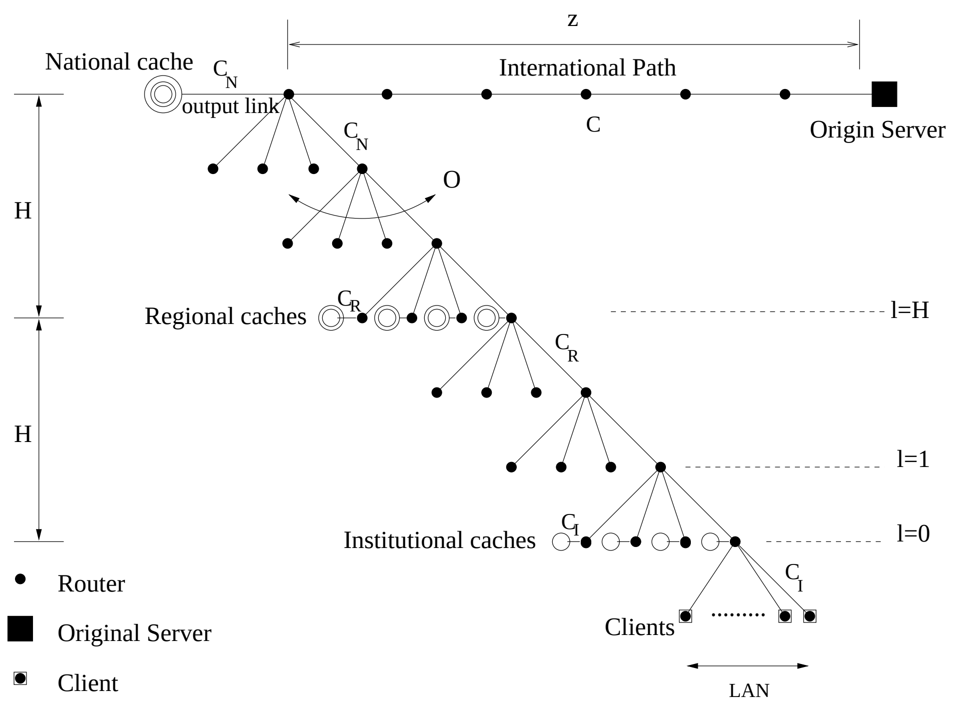


Figure 2.4: The tree model. Caching placement.

For simplicity we assume that clients' local caches are disabled. (We could include local client caches in the model, but they would only complicate the analysis without changing the main conclusions.) The **hit rate** is the percentage of requests satisfied by the caching hierarchy for all documents. The cumulative hit rate at a cache level or below for the institutional, the regional, and the national caches is given by HIT_I, HIT_R, HIT_N. To keep the analysis simple we assumed some typical values for the hit rates such as $HIT_I = 0.5$, $HIT_R = 0.6$, $HIT_N = 0.7$ [85] [79]. In Section 4.4.1 we derive analytical expressions for the hit rates at different caching levels.

A caching hierarchy cannot satisfy all requests arriving to it. Requests not satisfied on the caching hierarchy are called **misses**. We assume that each cache has infinite storage capacity since it is becoming very common to have caches with huge effective storage capacities. We also ignore non-cacheable misses (e.g. dynamic documents generated from cgi scripts), as they do not impact significantly the main conclusions of this chapter. In our analysis we consider hierarchical caching, however, the caching hierarchy can be easily replaced by a caching infrastructure with a mesh configuration as described in [93]. Thus, along this thesis we will use the terms caching hierarchy and caching infrastructure interchangeable.

2.2.3 CMP Model

For CMP we assume that the same hierarchical caching infrastructure is in place, and that most of the documents are distributed with the caching infrastructure. However, the N_{HC} hot-changing documents are distributed with CMP.

For the multicasting, we assume the origin server to be connected to the clients via a core based tree [14, 29]. The server sends to the core, which is the root for a shortest path tree [32], where a receiver is connected to the core via a shortest path through the network. Let μ_{cmp} be the multicast transmission rate for a single document. In Section 2.5.2 we shall show how μ_{cmp} can be calculated.

2.3 Latency Analysis for CMP

In this section we model the expected latency to distribute a hot-changing document by CMP. The average end-to-end latency, denoted by T, has two parts:

1. $E_{cmp}[T_c]$ the connection time. This is the time to join the multicast tree with the document.

2. $E_{cmp}[T_t]$ the transmission time. This is the time to transmit the document, which is equal to the document length S divided by the multicast transmission rate, $E_{cmp}[T_t] = S/\mu_{cmp}$.

Thus, the average latency for a hot-changing document when distributed by CMP is

$$E_{cmp}[T] = E_{cmp}[T_c] + S/\mu_{cmp}$$

We now proceed to calculate $E_{cmp}[T_c]$. We assume that the receiver knows the multicast address associated with the Web document, and the paths are symmetric with respect to delay, loss, etc. The connection time is measured up to the point in time where the receiver gets the first bit of the document. Let L be a random variable denoting the number of links traversed to meet the multicast tree. Because we are assuming a propagation delay of d seconds in each direction, the connection time is

$$E_{cmp}[T_c] = 2d \cdot E_{cmp}[L] \tag{2.1}$$

The expected number of traversed links that a join needs to travel in order to meet the multicast tree is:

$$E_{cmp}[L] \;=\; \sum_{l=0}^{2H+z} (l+1) \cdot P(L=l),$$

where we consider that there is one link that connects the client with the institutional ISP.

To obtain $P(L=l)$ we use:

$$P(L=l) = \begin{cases} P(L \geq l) - P(L \geq l+1) & 1 \leq l < 2H+z \\ \\ P(L \geq l) & l = 2H+z \end{cases} \tag{2.2}$$

Note that $P(L \geq l)$ is the probability that a new join meets the multicast tree carrying the document at level l or higher, but not before. Clearly $P(L \geq 0) = 1$.

Now consider the sub-tree rooted at level $l-1$. Requests for a document are generated at a rate λ_{l-1} within this sub-tree. Each request causes the multicast tree to extend on that subtree towards the requesting receiver (if the tree is not already there). Each receiver keeps the tree extended during the transmission time of the document, S/μ_{cmp} The number of requests being serviced is the number of customers in an $M/D/\infty/\infty$ queue with arrival rate λ_{l-1} and service time S/μ_{cmp}. The probability that a join has to travel more than $l-1$ levels is the probability that there are no customers in the $M/D/\infty/\infty$ queue at level $l-1$:

$$P(L \geq l) = \begin{cases} 1 & l = 0 \\ \\ e^{(-\lambda_{l-1} \cdot S/\mu_{cmp})} & l \geq 1 \end{cases} \tag{2.3}$$

Having calculated the distribution of L, we can determine $E_{cmp}[T_c]$ and therefore the total average latency, $E_{cmp}[T]$:

$$E_{cmp}[T] \;=\; 2d \cdot \sum_{l=0}^{2H+z} (l+1) \cdot [e^{(-\lambda_{l-1} \cdot S/\mu_{cmp})} - e^{(-\lambda_l \cdot S/\mu_{cmp})}] + \frac{S}{\mu_{cmp}}$$

2.4 Latency Analysis for Hierarchical Caching

In this section we determine the average latency for a hot-changing document which is updated periodically every Δ seconds. The average latency for hierarchical caching has two components:

1. $E_{cache}[T_c]$, the time to connect to the document. This is the time to find the document in the nearest server (cache or origin) plus the round-trip times for establishing the TCP connections.

2. $E_{cache}[T_t]$, the transmission time for the document. This is the time to transmit the document from server/cache to client.

We have

$$E_{cache}[T] = E_{cache}[T_c] + E_{cache}[T_t].$$

2.4.1 Connection Time

We now determine $E_{cache}[T_c]$. Let L denote the number of links traversed to find the document. L is a random variable which takes values in $\{0, H, 2H, 2H+z\}$. We model

$$E_{cache}[T_c] = 4d \cdot E_{cache}[L].$$

The rationale for the $4d$ term is as follows. In a caching hierarchy a TCP connection is opened between every caching level before starting the transmission of the Web document. In a multicast distribution the delivery is open-loop and there is no previous connection setup. A TCP connection uses a three-way handshake protocol that doubles the number of links traversed compared to a multicast distribution. We assume that the operating system in the cache gives priority at establishing TCP connections versus TCP connections that are already established.

We now proceed to calculate the distribution of L for the caching hierarchy. To obtain $P(L = l)$ we use

$$P(L = l) = \begin{cases} 1 - P(L \geq H) & l = 0 \\ \\ P(L \geq H) - P(L \geq 2H) & l = H \\ \\ P(L \geq 2H) - P(L \geq 2H + z) & l = 2H \\ \\ P(L \geq 2H + z) & l = 2H + z \\ \end{cases} \tag{2.4}$$

Note that $P(L \geq l)$ is the probability that the document is present at level l or higher, but not before. Consider the term $P(L \geq H)$. This is the probability that a request from an institution does not find the document in the institutional cache. Let w denote the time in the interval $[0, \Delta]$ at which a request occurs. The random variable w is uniformly distributed over the interval. Thus

$$P(L \geq H) = \frac{1}{\Delta} \int_0^{\Delta} P(L \geq H | w = \tau) d\tau$$

Now $P(L \geq H | w = \tau)$ is the probability that no request from the same institution arrives in the interval $[0, \tau]$, i.e.

$$P(L \geq H | w = \tau) = e^{-\lambda_I \cdot \tau}$$

Combining these two formulas and integrating gives

$$P(L \geq H) = \frac{1}{\lambda_I \cdot \Delta}(1 - e^{-\lambda_I \cdot \Delta})$$

Similarly

$$P(L \geq 2H) = \frac{1}{O^H \cdot \lambda_I \cdot \Delta}(1 - e^{-O^H \lambda_I \Delta})$$

$$P(L \geq 2H + z) = \frac{1}{O^{2h} \cdot \lambda_I \cdot \Delta}(1 - e^{-O^{2H}\lambda_I\Delta})$$

Notice that $\lambda_I\Delta$ is the expected number of requests from an institution in an update period. As $\lambda_I\Delta \to \infty$, the above three probabilities approach zero because with high probability the document is in the institutional cache. On the other hand, as $\lambda_I\Delta \to 0$, the above probabilities approach one because with high probability the document is only available at the origin server.

2.4.2 Transmission Time

The transmission time is the time needed to send the document from server to client once the TCP connection is in place. We make the realistic assumption that the caches operate in a cut-through mode rather than a store-and-forward mode, i.e., when a cache begins to receive a document it immediately transmits the document to the subsequent cache (or client) while the document is being received.

The transmission time depends on L, the closest level with a copy of the document:

$$E_{cache}[T_t] = \sum_{l \in \{0,H,2H,2H+z\}} E_{cache}[T_t | L = l] \cdot P(L = l)$$

Recall that the distribution of L is given in previous subsection. We now proceed to calculate $E_{cache}[T_t | L = l]$. To this end, we determine the aggregate request arrival rate for each of the caches. The request arrival rate for the instituional, regional, and national caches is denoted by β_I, β_R and β_N, respectively. The average request rate for all documents arriving to a cache at level l is filtered by the hits at lower caches:

$$\beta_I = \beta_{LAN} \quad l = 0$$

$$\beta_R = O^H \beta_{LAN}(1 - HIT_I) \quad l = H$$

$$\beta_N = O^{2H} \beta_{LAN}(1 - HIT_R) \quad l = 2H$$

In order to estimate $E_{cache}[T_t|L = l]$ we use a simple M/D/1 queue to model the queuing delays on the ouput links of the instutional, regional and national networks; see Figure 2.5.

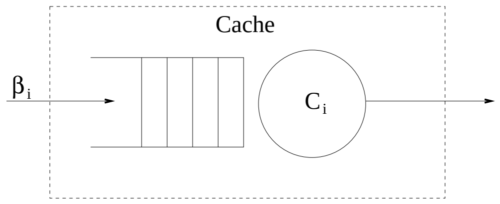


Figure 2.5: Queueing model for the load on the caches.

Let D_j be the delay at a given level of the caching hierarchy. This delay accounts for the queueing time plus the service time S/C_j. The M/D/1 model gives:

$$D_j = \frac{S}{C_j - \beta_j S} \cdot (1 - \frac{\beta_j S}{2C_j}) \quad j \in \{I, R, N\}$$

We assume that the delay experienced by a client when a document is hit a level l of a caching hierarchy is given by the delay at level l. Delays on lower caching levels than l are neglected. Thus, $E_{cache}[T_t|L = l]$ is given by:

$$E_{cache}[T_t|L = l] = \begin{cases} D_I & l = 0 \\ D_R & l = H \\ D_N & l = 2H \\ \dfrac{S}{\mu_{cache}} & l = 2H + z \end{cases} \tag{2.5}$$

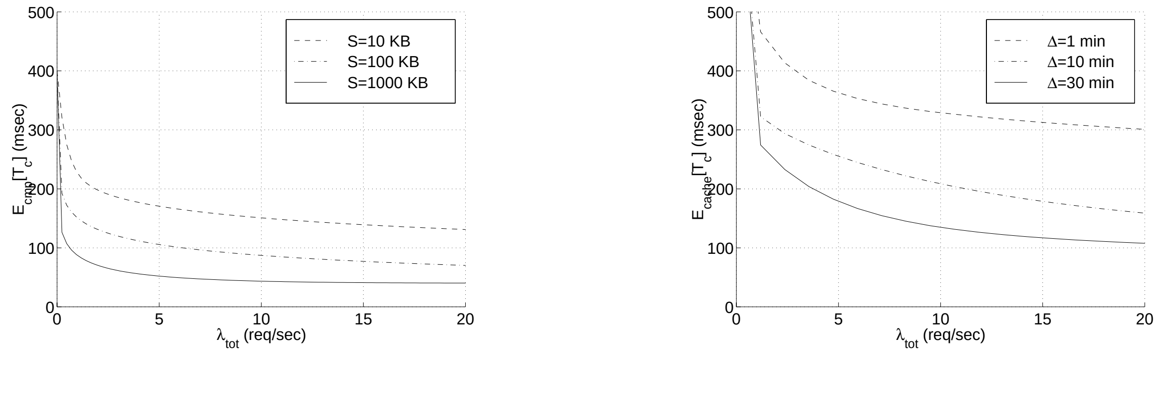
where μ_{cache} is the caching transmission rate for a single document on the International Path. In Section 2.5.2 we consider in more detail how to calculate μ_{cache}.

2.5 Latency Comparison

The following parameteres will be fixed for the remainder of the chapter, except when stated differently. The network is modelled with $O = 4$ as nodal outdegree of the multicast tree; $H = 3$ as the distance between cache hierarchy levels, yielding $O^H = 64$ regional caches and $O^{2H} = 4096$ institutional caches; $z = H = 3$ as the distance in the International Path.

2.5.1 Connection Time

The connection time for a CMP and a cache distribution depends on how close the document is to the receivers at every moment. In a CMP distribution, a document is at a certain level of the multicast tree for a time equal to the transmission time S/μ_{cmp} regardless of its update period Δ. In a caching distribution, a document is at a certain level of the caching hierarchy for a time equal to the update period Δ regardless of its document size S (given an infinite cache size). If the cache space is limited a document can be also removed from a cache due to space constraints.



(a) Multicast (b) Caching

Figure 2.6: Connection time $E_{cmp}[T_c]$ for a multicast and connection time $E_{cache}[T_c]$ for a caching distribution, depending on the total request rate for different update periods Δ and document sizes S. $\mu_{cmp} = 1$ KB/sec. $d = 20$ msec.

Figure 2.6(a) shows the Connection time for a CMP distribution $E_{cmp}[T_c]$ depending on different document sizes S and on the total request rate λ_{tot}. The values for the parameters μ_{cmp} and d are taken from recent caching studies [85] [79] [13]. We model different transmission times S/μ_{cmp} for a document by varying the document size. When the number of requests is very small, it is very likely that a join has to travel all the way to the original server in order to meet the multicast tree. An arriving request can not share any branch of the multicast tree built for past requests because it is already shrunk. When the number of requests is high a new request will meet the multicast tree at a lower level. For very popular documents regardless of its update period Δ the multicast tree is always very close to the receivers.

As we see in Figure 2.6(a), the connection time of a CMP distribution clearly depends on the document size S. For very small documents, the transmission time is very small. The tree shrinks very frequently, reducing the probability of meeting the tree at a low level. For larger documents the tree is kept extended for a longer time reducing the connection time of new requests.

Figure 2.6(b) shows the connection time for a caching distribution: $E_{cache}[T_c]$ depends on λ_{tot} and the update period Δ. We see that if the document is rarely requested, the average number of travelled links needed to meet a cache with an up-to-date document is high. For high request rates, a newly arriving request meets the up-to-date document at a closer caching level.

The main reasons why the connection time on a CMP distribution is shorter than that on a caching distribution are the following: 1) A multicast tree has a higher degree of granularity than a caching hierarchy. If a document is not hit at level l it can be hit at level $l + 1$. However, on a caching hierarchy if a document is not hit at level l, the closest level where the document can be hit is at level $l + H$. 2) When a document is very popular the multicast tree is always very close to the receivers independently on how fast the document changes. In a caching distribution even if a document is very popular the number of links traversed to hit the document is very much determined by the update period Δ. 3) A multicast distribution is an open-loop distribution where no connection is established. A caching distribution uses

TCP to previously establish a connection, which adds a factor of two to the connection time.

2.5.2 Transmission Time

In order to provide results for $E_{cmp}[T_t]$ and $E_{cache}[T_t]$, we need to calculate μ_{cmp} and μ_{cache}. First, we calculate the transmission rate μ_{cmp} for a hot-changing document in a CMP distribution . Then, we argue that the transmission rate μ_{cache} (equation 2.5) for a hot-changing document in a caching distribution when the document is hit at the original server is $\mu_{cache} = \mu_{cmp}$. The transmission rate μ_{cmp} for a hot-changing document is determined by the minimum transmission rate at any level of the network. The link with the most traffic and the lowest capacity is the international link. Therefore, the international link is the bottleneck for the end-to-end multicast transmission. In this calculation, we assume that there is at least one interested receiver for each of the N_{HC} hot-changing documents at every moment. In this situation, a multicast distribution needs to continuously send the N_{HC} documents from the original server to all institutions. The available end-to-end CMP transmission rate for the hot-changing documents is equal to the capacity C on the international path minus the capacity needed for the background traffic that is not satisfied by the caching hierarchy. If this capacity is equally shared by the N_{HC} hot-changing documents, the CMP transmission rate for a hot-changing document is given by:

$$\mu_{cmp} = \frac{C - \beta_{LAN}^{B} O^{2H} (1 - Hit_N) \cdot S}{N_{HC}}$$

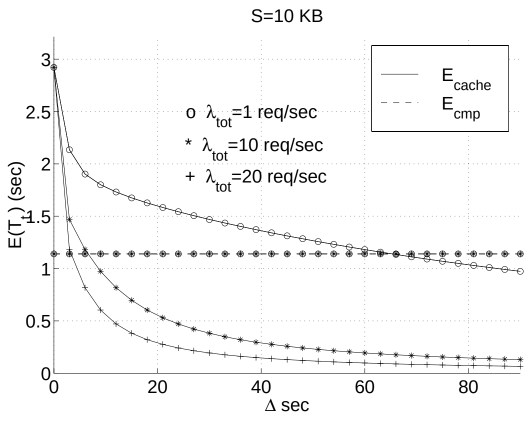
Next, we argue that $\mu_{cmp} = \mu_{cache}$. A CMP distribution is continuously sending the N_{HC} hot-changing documents at a rate μ_{cmp}. If the hot-changing documents change so frequently that every document request sees a document update, a caching distribution resembles to a CMP distribution in the sense that the N_{HC} hot-changing documents need also to be continuously delivered from the original server. In this situation the rate μ_{cache} at the International Path on a caching distribution is equal to the end-to-end CMP rate $\mu_{cache} = \mu_{cmp}$. When several requests see an unmodified document, only one copy of the document is sent through the international path to the caching hierarchy every period Δ. The rest of the requests inside that period Δ are satisfied from local copies at lower level caches. In this situation the available caching transmission rate μ_{cache} for one hot-changing document through the international path is higher than the CMP rate μ_{cmp}. Therefore considering $\mu_{cache} = \mu_{cmp}$ is being pessimistic for the transmission time in a caching distribution through the international path.

2.5.3 Numerical Analysis

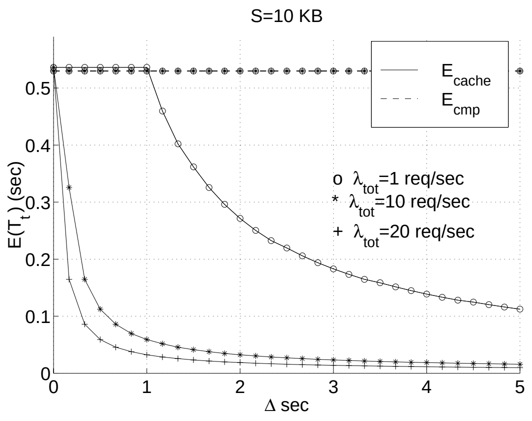
We pick some typical values for the different parameters in the model. We take $\beta^{B}_{LAN}/\beta^{HC}_{LAN} = 10$, meaning that the request rate for hot-changing documents is ten times lower than the request rate for the rest of the traffic. This ratio may vary if more documents appear to be frequently changing. We have also chosen some indicative values for the link capacities at the different hierarchy levels: $C_I = 100$ Mbps, $C_R = 45$ Mbps, $C_N = 45$ Mbps, $C = 34$ Mbps.

We analyze two different scenarios: (1) The access link of the national cache is being used close to its full capacity and therefore the queuing delays on the national cache increase the caching latency; (2) The access link of the national cache is not being used close to its maximum capacity and the bottleneck for a caching distribution is the limited bandwidth of International Path. For a CMP distribution the bottleneck is always in the International Path. We model these two different scenarios by varying β_N. Choosing $\beta_N S = 0.9C_N$ we can model the first scenario. Decreasing β_N to $\beta_N S = 0.6C_N$, we can model the second scenario. Given the two scenarios we calculate the transmission and total latency for a hot-changing document that is distributed through a caching or a CMP model. Varying the update period Δ and the total request rate λ_{tot}, we determine when caching has lower latency then CMP.

In Figure 2.7 we plot the transmission time for a document on a caching and a CMP distribution depending on the update period of the document for the two different scenarios. From Figure 2.7(a) we see that if the bottleneck for a caching distribution is given by the limited access link of the national cache, a caching distribution of frequently-changing documents has higher transmission times than a CMP distribution. However, when the bottleneck for both a CMP and a caching distribution is placed on the International Path (Figure 2.7(b)),



(a) The bottleneck is given by the limited bandwidth on the access link of the national cache. $\beta_N S = 0.9 \cdot C_N$



(b) The bottleneck is given by the limited bandwidth on the International Path. $\beta_N S = 0.6 \cdot C_N$

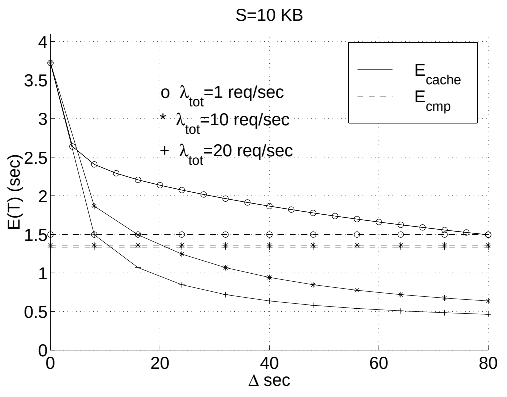
Figure 2.7: Cache and CMP transmission time T_t for a hot-changing document for different Δ. $N_{HC} = 100$, $C_I = 100$ Mbps, $C_R = 45$ Mbps, $C_N = 45$ Mbps, $C = 34$ Mbps. $S = 10$ KB.

a frequently-changing document will have the same transmission time in a caching distribution and a CMP distribution. When the document does not change more frequently than a few seconds 2.7(b) or a few tens of seconds 2.7(a) a caching distribution always has a lower transmission time than a CMP distribution.

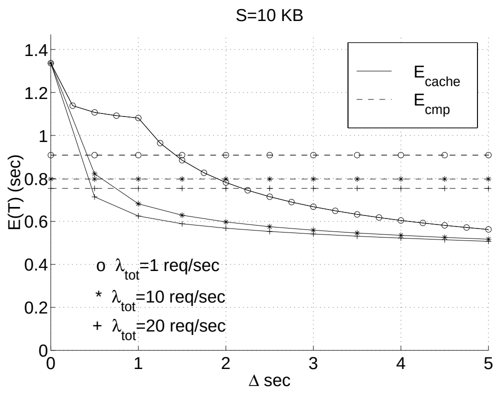
2.5.4 Total Latency T

In Figure 2.8 we plot the total latency T comprising the connection time T_c and the transmission time T_t for CMP and caching. We have also considered both bottleneck scenarios described on the previous section.

Comparing Figure 2.7 and Figure 2.8 we see that adding the connection time T_c to the transmission time T_t increases the latency differences between a caching distribution and a CMP distribution for a frequently-changing document. Additionally, we see that the curves are displaced to the right increasing the point where caching is better than CMP.



(a) The bottleneck is given by the limited bandwidth on the access link of the national cache. $\beta_N S = 0.9 \cdot C_N$



(b) The bottleneck is given by the limited bandwidth on the International Path. $\beta_N S = 0.6 \cdot C_N$

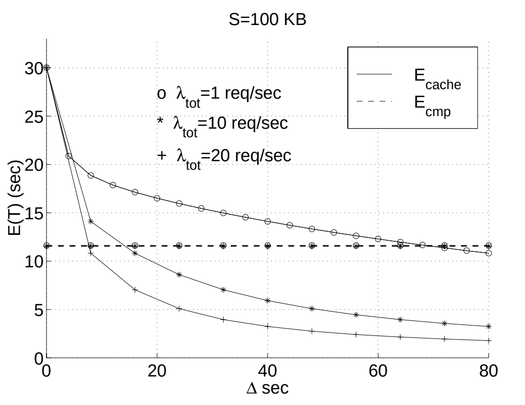
Figure 2.8: Cache and CMP Total Latency T for a hot-changing document for different Δ. $N_{HC} = 100$, $C_I = 100$ Mbps, $C_R = 45$ Mbps, $C_N = 45$ Mbps, $C = 34$ Mbps. $S = 10$ KB.

From Figure 2.8(a) we observe that if the access link of the national cache is very congested and the documents change faster than several tens of seconds a CMP distribution has a lower total latency than a caching distribution. When the access link of the national cache is less congested (Figure 2.8(b)), the value of Δ where caching gives lower latencies than CMP gets smaller.

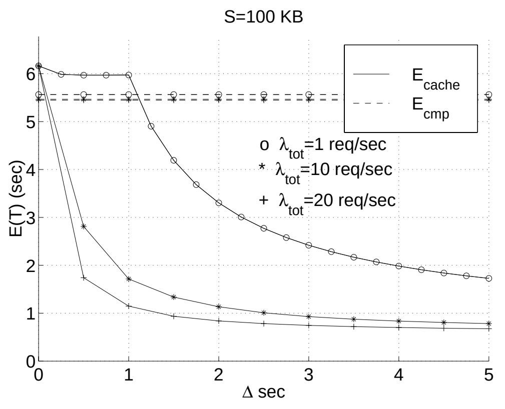
The total latency of a CMP distribution is almost insensitive to changes in the total request rate for a single document λ_{tot}. Only the connection time of a CMP distribution depends on λ_{tot}; the transmission time of a CMP distribution is independent λ_{tot}. The latency in a caching distribution has a higher dependency on the popularity of the document (given by λ_{tot}) than in a CMP distribution. Both, the connection time and the transmission time of a caching distribution depend on λ_{tot}. For higher λ_{tot} the document is found at lower levels which have higher capacities. The less popular the document, the higher is the latency time to retrieve a document in a caching hierarchy.

Note that we only consider documents that are very popular, which is why λ_{tot} takes very high values. We claim that non-popular documents should not be sent via CMP because the bandwidth savings are not significant while there is significant overhead in maintaining many multicast trees.

In Figure 2.9 we plot the transmission time for a larger document $S = 100$ KB. We see that when the document size increases the the latency differences between a caching distribution and a CMP distribution get reduced for those values of Δ where a CMP distribution is better than a caching distribution. This is because for large document sizes, the connection time contributes less to the total latency than the transmission time. A surprising result is that varying the document sizes, the value of Δ for which a CMP distribution has lower latency than a caching distribution varies only slightly. This is an interesting result because it suggests that the point at which the CMP distribution is better than the caching distribution does not greatly depend on the document size.



(a) The bottleneck is given by the limited bandwidth on the access link of the national cache. $\beta_N S = 0.9 \cdot C_N$



(b) The bottleneck is given by the limited bandwidth on the International Path. $\beta_N S = 0.6 \cdot C_N$

Figure 2.9: Cache and CMP Total Latency T for a hot-changing document for different Δ. $N_{HC} = 100$. $C_I = 100$ Mbps, $C_R = 45$ Mbps, $C_N = 45$ Mbps, $C = 34$ Mbps. $S = 100$ KB.

2.5.5 Random Updates

For periodic updates, the caches do not need to poll the source to check the document's consistency. Indeed, the document is removed from the caches after a period Δ; if the document is in the cache, then the document is up-to-date. For random document updates, if caches want to provide strong consistency to the receivers, the caches need to poll the server on every request. In this case, the number of links traversed to establish a TCP connection is equal to the number of links between the receivers and the origin server, i.e., $E_{cache}[L] = 10$. For a CMP distribution, the connection time does not depend on the update period Δ of a document. Looking at Figure 2.10 we observe that a CMP distribution has always lower connection times than a caching distribution when strong document consistency is required.

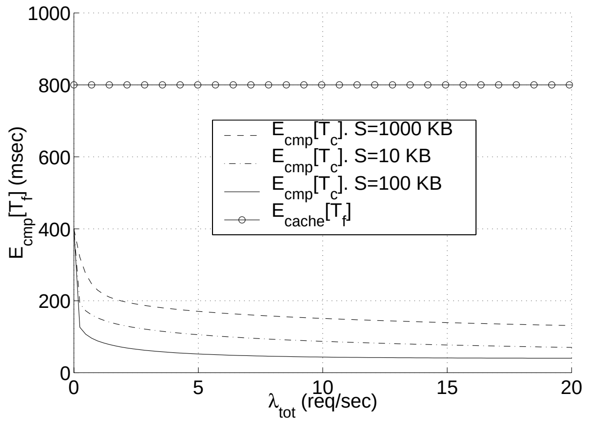


Figure 2.10: Connection time for a CMP and a caching distribution $E_{cmp}[T_c]$, $E_{cache}[T_c]$ when documents are randomly updated.

Assuming that the update period t is exponentially distributed with average Δ, $f(t) = \frac{1}{\Delta}e^{-t/\Delta}$, then the distribution of L is:

$$P(L = l) = \int_0^\infty P(L = l|t)f(t)dt, \tag{2.6}$$

where $P(L = l|t)$ is given by equation 2.4. For $l = 0$ we have that

$$P(L = 0|t) = 1 - P(L \geq H|t) = 1 + \frac{e^{-\lambda_I \cdot t} - 1}{\lambda_I \cdot t}. \tag{2.7}$$

Combining equations(2.6) and (2.7) we obtain that

$$P(L=0) = 1 + \frac{1}{\lambda_I \cdot \Delta} \cdot ln(\frac{1}{1+\lambda_I \cdot \Delta}). \tag{2.8}$$

The distribution of L for other caching levels can be obtained in a similar way.

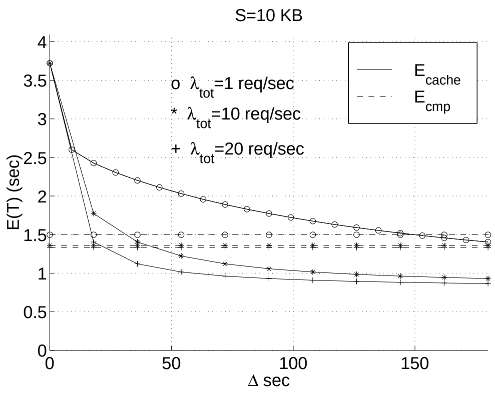
In Figure 2.11 we plot the total latency $E_{cmp}[T]$ and $E_{cache}[T]$ for random updates given that the access link of the national cache is a bottleneck and given that the bottleneck is on the international path. From Figure 2.11(a) we see that for random updates the total latency for a caching distribution increases compared to the case where the documents change periodically (Figure 2.8(a)). Additionally the value of Δ for which CMP has lower latencies that hierarchical caching increases.

For small document sizes $S = 10$ KB, the connection time becomes very relevant especially when the network is not congested. In Figure 2.11(b) the access link of the national cache is not congested and the connection time dominates over the transmission time. Given that for random updates the connection time is always higher for a caching distribution than for a CMP distribution (Figure 2.10) the total latency of a caching distribution is higher than the total latency of a CMP distribution even for higher values of Δ.

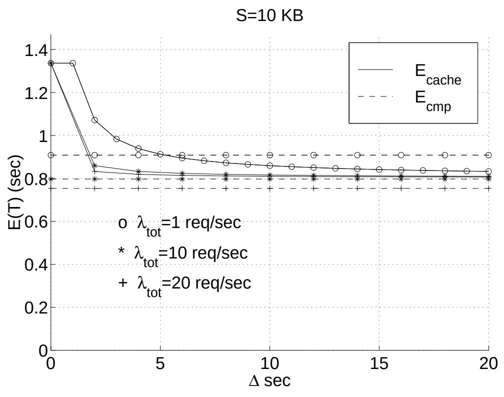
Therefore, in the case that a document is randomly updated a caching distribution needs to poll every time the server, increasing the total latency over that in a CMP distribution. One way to avoid having caches check for the document's freshness is to use a source-initiated invalidation scheme [58]. The origin server sends invalidation messages to communicate the caches that a certain document has expired. The caches do not need to worry about the consistency of the document; if the document is in the cache, it is up-to-date; if the document is not in the cache, it is because no one has yet asked for the last version of the document (see Chapter 5 for a more detailed discussion on different consistency mechanisms).

2.6 Bandwidth Comparison

Institutional networks are connected to regional ISPs via access links, and regional ISPs are connected to national ISPs via access links (see Figure 2.12). While end users are concerned



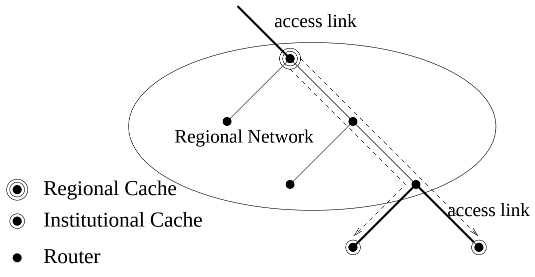
(a) The bottleneck is given by the limited bandwidth on the access link of the national cache. $\beta_N S = 0.9 \cdot C_N$



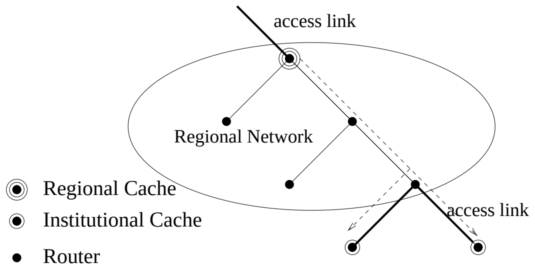
(b) The bottleneck is given by the limited bandwidth on the International Path. $\beta_N S = 0.6 \cdot C_N$

Figure 2.11: Total Latency T on a multicast distribution and a caching distribution when documents are randomly updated.

with the retrieval latency, ISPs are mainly concerned with bandwidth usage inside their network and bandwidth usage in their access links. In this section we calculate i) the bandwidth usage in the access links and ii) the bandwidth usage inside every ISP.



(a)



(b)

Figure 2.12: Caching distribution (left) and Multicast distribution (right).

When a popular Web document expires frequently, a caching hierarchy resembles a CMP distribution in the sense that a new document update is continuously transmitted from the origin server to the receivers. However, within an ISP a caching distribution requires more network bandwidth than a multicast distribution because the communication between the different cache levels is done via unicast and not via multicast (Figure 2.12). When a popular document does not change frequently, a CMP distribution uses more bandwidth (in access link and within ISP) than a cache distribution because a CMP distribution sends the same document over and over the same links while a cache distribution only sends the document once every period Δ. We now develop a model that quantifies these observations.

2.6.1 Bandwidth in the Access Link

We first calculate the **bandwidth usage** in the access link for a cache distribution, we then calculate the same bandwidth usage for a CMP distribution. We do the analysis for the access link than joins the institutional network to the regional network, and for the access link that joins the regional network to the national network.

First consider a caching hierarchy. The average bandwidth usage by a hot-changing document in the access link that connects the institutional network to the regional network is given by:

$$BW_{cache} = \frac{S}{\Delta} \cdot [1 - exp(-\lambda_I \cdot \Delta)]$$

where $1 - exp(-\lambda_I \cdot \Delta)$ is the probability that the document is sent across the institutional-regional access link in a period Δ. For the regional-national access link this probability is given by $1 - exp(-\lambda_I \cdot O^H \cdot \Delta)$, which is higher than the probability for the institutional-regional access link because the number of receivers that are satisfied through the regional-national link is higher.

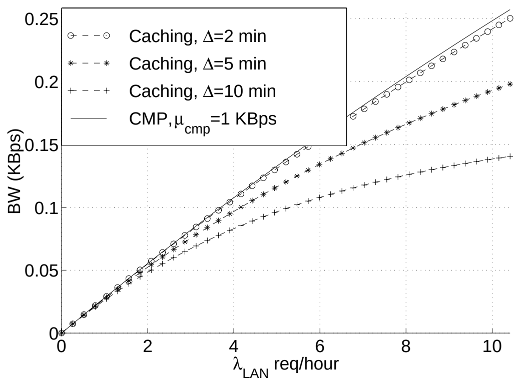
For a CMP distribution we assume that the multicast server is sending at a constant rate μ_{cmp}. The average bandwidth usage in the access link depends on the probability that the multicast tree is extended through that access link. For the institutional-regional access link the bandwidth is given by:

$$BW_{cmp} = \frac{S}{\Delta} \cdot P(L = 1)$$

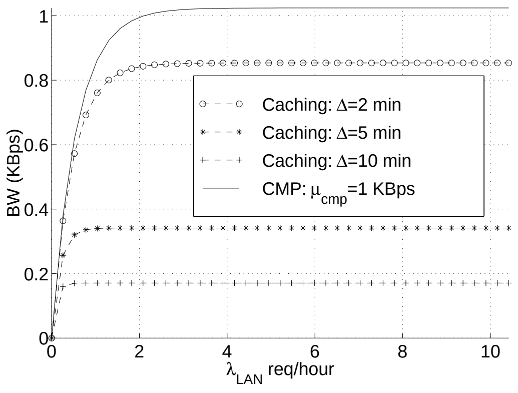
where $P(L = 0)$ is the probability that the multicast tree is extended to level $l = 0$. From equation 2.3:

$$P(L = 0) = 1 - exp(-\lambda_I \cdot S/\mu_{cmp})$$

For the regional-national access link the probability that the multicast tree is extended through level $l = H$ is given by $1 - exp(-\lambda_I \cdot O^{H-1} \cdot S/\mu_{cmp})$.



(a) Bandwidth usage in the institutional-regional access link.



(b) Bandwidth usage in the regional-national access link.

Figure 2.13: Bandwidth used by a caching distribution and a CMP distribution in the access link of an institutional network and a regional network. $S = 100$ KB. $\mu_{cmp} = 1$ KBps.

Figure 2.13 shows the access link bandwidth usage by a CMP distribution and a cache distribution depending on the request rate from an institution. We take $S = 100$ KB and $\mu_{cmp} = 1$ KBps as illustrative values. We see that when the Web document changes every 2 minutes or faster, both a CMP distribution and a cache distribution use similar bandwidth in the access link. However, when the Web document changes less often a CMP distribution sends more copies of the same document through the access link than a cache distribution, resulting in a higher bandwidth usage. The bandwidth differences between a CMP distribution and a caching distribution are higher in the access link that joins the regional and the national networks than in the access link that joins the institutional and the regional networks (see Figure 2.13).

2.6.2 Bandwidth in the ISP

In this section we calculate the bandwidth used inside one ISP. We consider an ISP at the regional level and an ISP at the national level.

First consider a cache distribution. The bandwidth usage by a cache distribution inside a regional ISP is given by:

$$BW_{cache} = (H-1) \cdot O^H \cdot [1 - exp(-\lambda_I \cdot \Delta)] \cdot \frac{S}{\Delta}$$

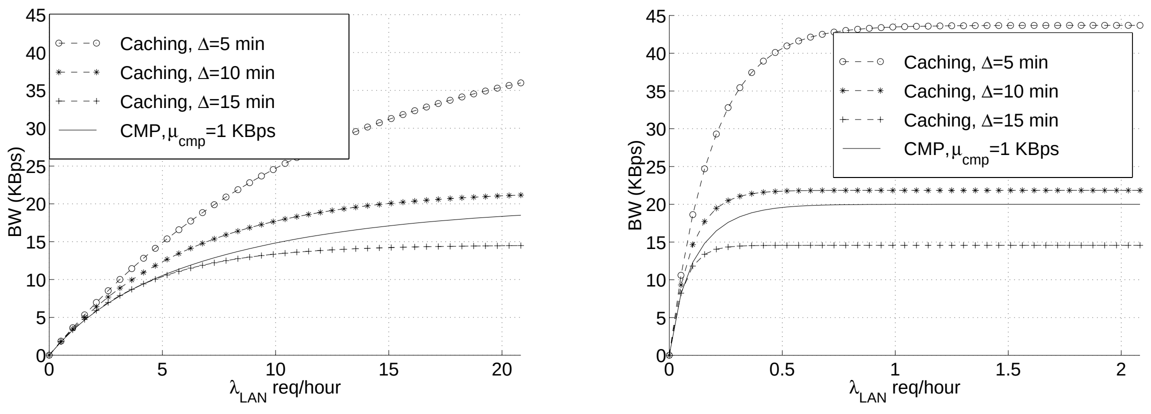
where $H-1$ is the number of network levels inside the regional ISP (without considering the access link) and $O^H \cdot [1-exp(-\lambda_I \cdot \Delta)]$ is the expected number of transmissions through any network level of the regional ISP. For a national ISP the expected number of transmissions through any of its network levels is given by $O^H \cdot [1 - exp(-\lambda_I \cdot O^H \cdot \Delta)]$

When a CMP distribution is used, the bandwidth in a regional ISP is given by:

$$BW_{cmp} = \mu_{cmp} \cdot \sum_{l=1}^{H-1} O^{H-l} \cdot P(L = l)$$

where $P(L = l)$ is the probability that the multicast tree is extended to level l (see equations 2.2 and 2.3). For the national ISP l takes values between $H+1$ and $2H-1$.

Figure 2.14 shows the total bandwidth usage inside a regional ISP and a national ISP when Web documents are distributed via hierarchical caching or via CMP. When a Web document changes every 10 minutes or faster, hierarchical caching uses more bandwidth than a CMP distribution. This is because hierarchical caching uses unicast transmissions between the different levels of the caching hierarchy while CMP shares all the common paths inside the network. However, if the document changes every 15 minutes or less then a CMP distribution uses more bandwidth than hierarchical caching because CMP is sending several copies of the same up-to-date document through the same links.



(a) Bandwidth usage in the regional ISP.

(b) Bandwidth usage in the national ISP.

Figure 2.14: Bandwidth used by a caching distribution and a CMP distribution inside a regional ISP and a national ISP. $S = 100$ KB. $\mu_{cmp} = 1$ KBps.

2.7 Conclusions

A caching hierarchy requires ISPs to invest in caches. But ISPs and Content distribution networks (e.g. Akamai [2]) are aggressively introducing caches throughout the world in order to reduce average latency and bandwidth usage. Furthermore, no fundamental changes need to be made to TCP/IP or routers in order to introduce caches into the Internet. On the other hand, multicast protocols operate at the network and transport layers. Because multicast requires fundamental changes in the Internet, widespread deployment of multicast in the Internet continues to be far way.

One of the applications of multicast is the distribution of Web documents. In this chapter we showed that unless a Web document changes very frequently, caching distribution gives lower latency and bandwidth usage than multicast. The reduced latency is mainly due to the fact that with multicast the transmission rate for a document is the bottleneck rate between server and client; for caching, the transmission rate is the bottleneck rate between the nearest cache with the document and the client. If a document changes very frequently, then CMP gives lower average latency than a cache hierarchy. This is because in a cache distribution when a document changes very frequently, an up-to-date document will rarely be available in a nearby cache. Additionally, requests for frequently changing documents will travel through top-level caches that can be congested, resulting in high queuing delays.

In this chapter we assumed a typical 3-tier caching hierarchy. However, our analysis can be easily extended to consider a caching hierarchy with a different number of cache tiers (See Chapter 3). Intuitively, the higher the number of cache tiers, the higher the probability to find an up-to-date document in a nearby cache, with high transmission rates. Therefore, when the number of cache tiers increases, the update period for which CMP performs better than hierarchical caching becomes even smaller. In this chapter we also assumed a lossless network. In the case of a lossy network where reliability is required, the relative *latency* performance of hierarchical caching and multicast would not be modified since reliability increases the mean number of transmissions per packet by a similar factor [76]. However, the relative *bandwidth* usage of reliable multicast versus reliable hierarchical caching would increase since multicast servers need to keep transmitting information through the International path until the most lossy client has reliably received the document.

Chapter 3

Further Advantages of Caching Infrastructures

In the previous chapter we showed that except for hot and frequently changing information, hierarchical caching has lower latencies and bandwidth usage than multicast. In this chapter we further study the performance advantages of hierarchical caching and extend our model to take into account different factors of a hierarchical caching such as the number of cache tiers or the disk requirements at the caches. In particular we study how a caching infrastructure can reduce the number of polls to the origin server when documents are frequently and randomly updated and clients must receive the most up-to-date information. In the case where popular information is frequently updated and changes randomly , the number of if-modified-since polls may overload top-level caches and origin servers. For this scenario we present new schemes for performing a push distribution using a caching infrastructure. We show how a caching hierarchy with a small number of tiers (i.e. three of four) can implement an application-level multicast distribution, mimicing the performance of IP-multicast. In the case where intra-domain multicast is available inside an ISP, we also explain how to combine a caching hierarchy with an intra-domain multicast distribution among cache tiers. This caching and multicast distribution scales very well with the number of documents, requiring only a small and fixed number of intra-domain static multicast trees.

3.1 Introduction

One of the main drawbacks of caching is that receivers may obtain stale documents. The current HTTP 1.0 protocol provides several mechanisms to keep cache consistency. Each document has a time-to-live (TTL) which is set by the server and indicates for how long the document will remain unchanged. If the server updates its content at fixed known intervals (i.e. periodically) the cache knows exactly when the document is fresh or stale without contacting the server. However, many times the origin server can not foresee when its documents are going to change and therefore it can not provide an accurate time-to-live. In this situation when the TTL expires the cache checks the document's consistency with the server. The cache can send an "if-modified-since" request to the origin server with a time-stamp. Upon reception, the server checks whether the document has been modified since the time-stamp. If so, the server returns the code "200" and the new copy; otherwise, the server returns the code "304", which stands for document unmodified.

The difficulty with the TTL approach is that it is often hard to assign the appropriate time-to-live of a document. If the value is too small, the server is burdened with many "if-modified-since" messages, even if the document is not changed. If the value is too large, the probability that the user sees a stale copy increases. The *adaptive TTL* approach tries to handle this problem by allowing the cache manager to assign a time-to-live value based on the observed lifetimes of the document. If caches are asked to deliver the most up-to-date copy to the receivers, a pooling every time mechanism must be used. The cache sends an "if-modified-since" request every time that a request for a document hits the cache [58].

Some new protocol headers concerning caching have been introduced in version 1.1 of HTTP [41], which is currently being deployed. This new headers provide significant improvement over the mechanisms used in HTTP 1.0. The new headers allow the origin servers to specify the maximum time that a document may be kept at the caches (max-age). Additionally, clients can specify the degree of staleness acceptable for a certain document, which relaxes the inflexible reload option that always polls the origin server.

In this chapter we study the impact of hierarchical caching in reducing the number of if-modified-since polls that arrive to the origin server when a polling every time mechanism is used and servers can set a certain max-age value. Using analytical models and trace-driven simulations we quantify the max-age interval that servers should set such that a caching hierarchy can filter out most of the if-modified-since polls to the origin server.

In the situation where documents change randomly or very fast, servers can not ensure even a small time interval during which a document is up-to-date and using a polling every time scheme generates too many poll requests to the origin server and does not scale. In this latter case, a push distribution should be implemented. In this chapter we consider how a caching hierarchy can implement a push distribution and present the benefits of using a caching hierarchy. Pushing through a caching hierarchy requires little state information at the origin server and the caches and requires little modifications to the existing infrastructure.

Though multicast may seem like a natural way of pushing information, when the number of documents delivered with multicast is high, it may lead to an unscalable IP-multicast architecture due to the high number of multicast trees. We present a new pushing scheme that combines a caching architecture and intra-domain multicast in between cache tiers. With a caching and multicast push, intermediate caches act as application level filters, only requiring a small and fixed number of static multicast trees, independently of the number of documents transmitted. In addition, the multicast trees are local to an ISP, thus, there is no need for a fully developed IP-multicast distribution.

The rest of this chapter is organized as follows. In Section 3.2 we extend our model from Section 2.2 so we can study the impact of other parameters in hierarchical caching. In Section 3.3 we calculate the total polling rate at the origin server with and without a caching hierarchy. We also calculate the disk space requirements at the caches. Section 3.4 describes the push scheme through a caching hierarchy and quantifies the state information required at the origin server and the caches. Section 3.5 studies a push scheme through a caching hierarchy combined with a multicast distribution among caching tiers for popular documents.

3.2 Extended Model

Next we extend the model presented in Section 2.2 to study the impact of an arbitrary number of cache tiers. As shown in Figure 3.1, we model the Internet as a hierarchy of ISPs, with each ISP having its own autonomous administration. Let $L + 1$ be the number of tiers of ISPs. We model each ISP in the top L tiers as a full O-ary tree of depth H [69], with a router at every tree node. In Figure 3.1, $L = 3$, $H = 3$, and $O = 4$. At the root of the top tier ISP there is the *origin server*. The level of the network tree l, ranges from $l = 0$ to $l = LH$. These assumptions imply that there are LH links from the origin server to each institutional ISP, and O^{LH-l} links between network level $l + 1$ and network level l.

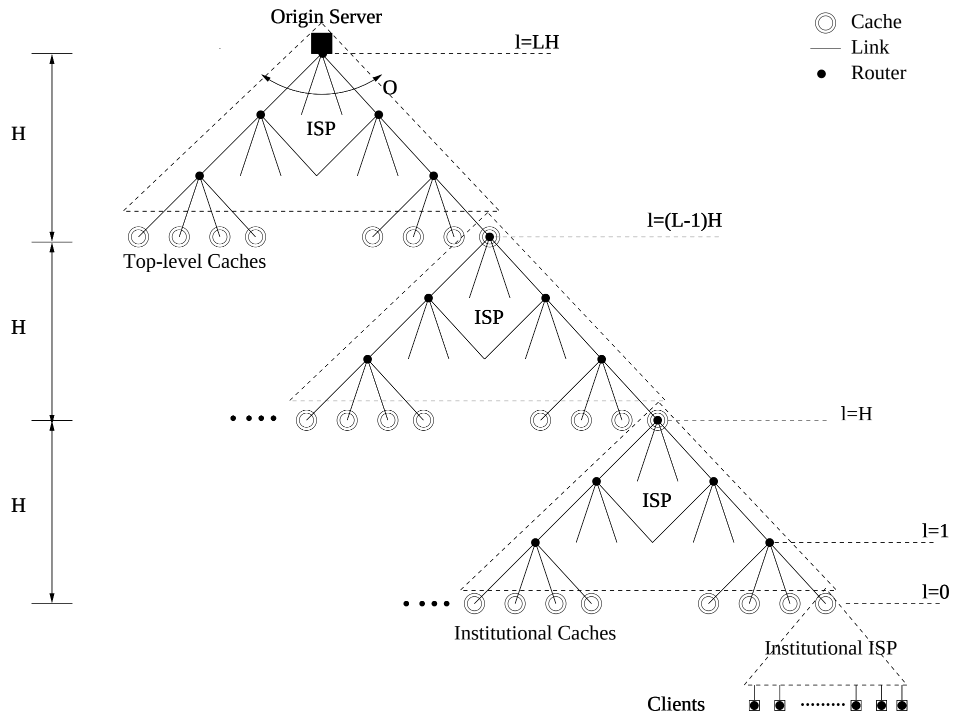


Figure 3.1: The tree model. Caching infrastructure.

To model a multiple tier caching hierarchy, we suppose that at the root of every ISP there is a cache, except at the root of the ISP in the top tier, where there is the origin server (see Figure 3.1). There are O^H *top-level* caches, and O^{LH} institutional caches. A document update needs to be received at an institutional ISP if at least one of the clients in the institutional ISP is interested in the document.

3.3 Polling Rate with Caching Infrastructure

Now, we study the **total polling rate** at the origin server in the scenario where caches do not know in advance when a document will expire and poll the origin server on each request to ensure that the client receives the most up-to-date copy. We consider the cases when i) there are only institutional caches at every institutional ISP and there are no intermediate caches, and when ii) there is a full caching hierarchy. The following analysis applies not only for a full O-ary tree network, but also for any network topology.

With always pull, clients poll the origin server on every request and fetch a new updated document version if the document has changed. We suppose that requests from multiple clients for the same document are Poisson distributed [45] [68] (in Section 3.3.1 we also consider other distributions for the document requests). From each institutional ISP the interarrival time between document requests is exponentially distributed with an average request rate λ_I (see Figure 3.2).

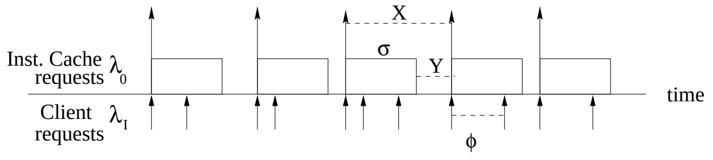


Figure 3.2: Polling rate at an institutional cache for a given max-age interval, σ.

We assume that origin servers can specify a max-age value, σ, (using the HTTP/1.1 cache-control headers [41]) during which caches do not poll the servers and assume that the document remains unchanged. Each document stored in a cache carries an *age*, ϕ. The age of a document is the period of time that a document copy has been in any cache of the caching infrastructure, that is, the time since the document was retrieved from the origin server. A cache can serve a document with age ϕ if the max-age value, σ, is greater than ϕ (Figure 3.2).

If origin servers wish to validate every request, they may set a max-age equal to zero, i.e. $\sigma = 0$, so that the caches always validate the document.

Some of the client requests that arrive at the institutional ISP with rate λ_I, will be filtered out by the institutional cache. Therefore, the rate λ_0 at which polling messages leave a institutional ISP, is smaller than λ_I. Referring to Figure 3.2, we now calculate λ_0. Since the time between requests is exponentially distributed and thus memoryless, the time Y between the end of a max-age interval and next poll request is also exponentially distributed with the same average request rate λ_I. Let X be the interarrival time between filtered poll requests. X is a random variable that is equal to the sum of the max-age interval and the random variable Y, that is, $X = \sigma + Y$. The expected value of X is $E[X] = \sigma + E[Y] = \sigma + \frac{1}{\lambda_I}$. Thus the expected poll rate λ_0 from an institutional cache is given by

$$\lambda_0 = \frac{1}{E[X]} = \frac{\lambda_I}{1 + \lambda_I \cdot \sigma} \ . \tag{3.1}$$

Figure 3.3 shows the request rate leaving an institutional cache λ_0 for different max-age intervals, σ. We see that if the origin server wants to validate all requests (i.e., $\sigma = 0$), the polling rate λ_0 is equal to the client request rate λ_I. However, as soon as a small max-age is set (i.e., $\sigma = 5$ minutes), the polling rate from an institutional cache is kept low even for very high client request rates.

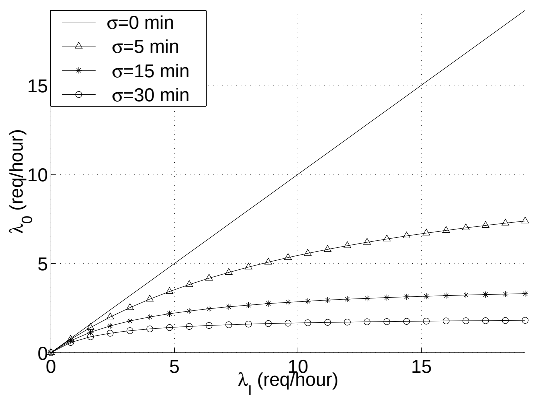


Figure 3.3: Polling rate λ_0 from an institutional cache depending on the clients request rate λ_I for different max-age intervals.

Next, we consider the case where there is a caching hierarchy. A caching hierarchy can filter out most of the if-modified-since polls to the origin server. To illustrate this, in Figure 3.4 we show the case of a two-level caching hierarchy (two institutional caches and one top-level cache). When institutional caches poll the origin server, only the first poll from any institutional cache is redirected by the top-level cache to the origin server. Further polls from any institutional cache during a max-age interval are filtered at the top-level cache (see Figure 3.4).

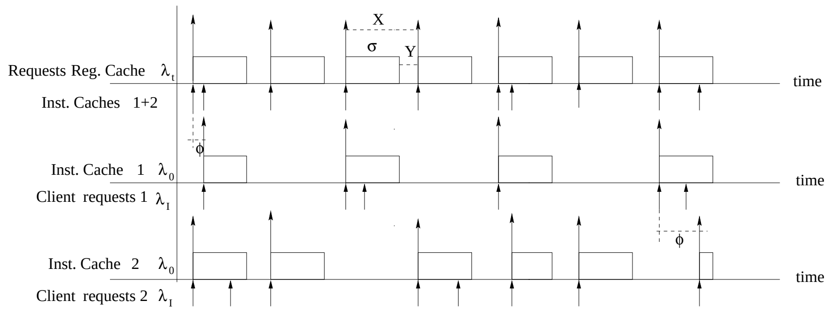


Figure 3.4: Request rate of a regional cache with two children institutional-caches

At a top-level cache, the interarrival time between polls to the origin server is given by $X_t = \sigma + Y_t$, where Y_t is exponentially distributed with an average request rate equal to $\lambda O^{(L-1)H}$. Thus, the polling rate generated by one top-level caches to the origin server is $\lambda_t = \frac{\lambda O^{(L-1)H}}{1+\lambda O^{(L-1)H}\sigma}$. The total polling rate λ_s^h at the origin server generated by the O^H top-level caches is given by

$$\lambda_s^h = \lambda_t O^H = \frac{\lambda_I O^{LH}}{1 + \lambda_I O^{(L-1)H}\sigma} \quad . \tag{3.2}$$

For very popular documents ($\lambda_I \to \infty$), the total polling rate at the origin server with no caching hierarchy approaches $\frac{1}{\sigma}O^{LH}$, since the polling rate from one institutional cache approaches $\frac{1}{\sigma}$. The polling rate at the origin server with a caching hierarchy approaches $\frac{1}{\sigma}O^H$. Thus a caching hierarchy reduces the polling rate at popular servers by a factor of $O^{(L-1)H}$, which is rather significant. To prefetch every document update in the caches with an automated pull, caches need to automatically poll the server at a rate $\lambda_I \geq \frac{1}{\sigma}$. For $\sigma = 0$, the polling rate from institutional caches tends to infinity and also a caching hierarchy does not filter any request, thus, resulting in a high burden for the origin server.

Next, we quantify the max-age value σ that servers must specify for an automated pull with a caching hierarchy to generate only few requests to the origin server. Figure 3.5 shows the total polling rate at the origin server for different values of max-age values with and without a caching hierarchy; the caching hierarchy has $L = 3$ tiers. When there is no caching

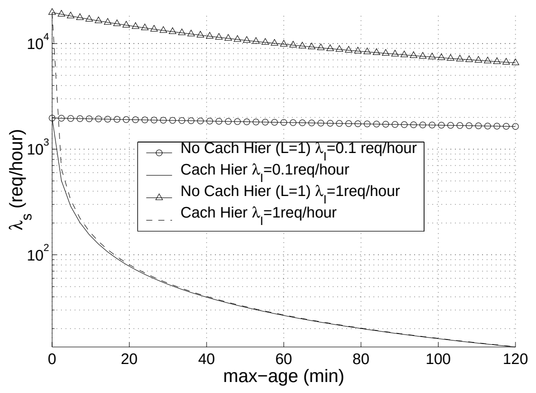


Figure 3.5: Total polling rate at the origin server for different max-age and different number of institutional caches. $H = O = 3$.

hierarchy the polling rate at the server stays high even for max-age values greater than few minutes. However, when there is a caching hierarchy, as soon as the server sets a small max-age value during which the document is not updated (i.e., few minutes), a caching hierarchy reduces the number of requests to the origin server by several orders of magnitude. Setting a small max-age of few minutes can be tolerated by most Web sites, including those offering stock tickers, news, or weather maps which may not be update their documents faster than once every few minutes due to the way the information is generated.

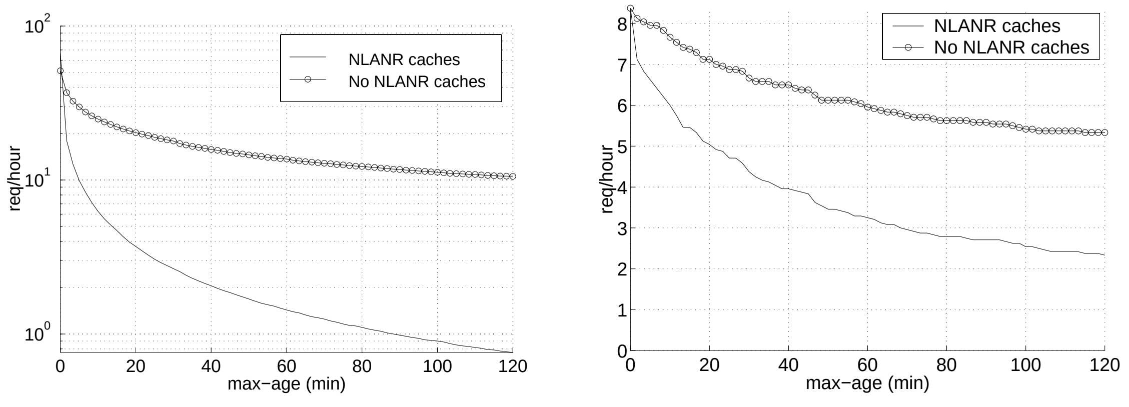
In figure 3.5 we have also varied the number of client requests λ_I, which may represent varying the popularity of a document or varying the polling rate of individual clients. We see that as λ_I increases (more popular documents or higher client polling rates), the total polling rate at the origin server increases significantly when there is no caching hierarchy. However, when there is a caching hierarchy, the total polling rate at the origin server is kept low even for high λ_I. We finally notice that the benefits of a caching hierarchy are less pronounced for small λ_I since there is a lower aggregation of requests at every parent cache. The higher the number of caches connected to a parent cache, the higher is the aggregation of requests at the parent cache, and therefore, the higher is the effect of a caching hierarchy in reducing the number of requests at the origin server.

3.3.1 Trace Driven Simulation

The Poisson request rate and the network topology used so far are not intended to be an accurate model of the real poll rate and network topology. In this section we present trace-driven simulations with real workloads and a real caching infrastructure to confirm our analytical model. For this purpose we have taken 10 days of logs during November 1999, from the four major parent caches, “bo”, “pb”, “sd”, and “uc” in the National Web Cache hierarchy by National Lab of Applied Network Research (NLANR) [6]. These caches are the top-level caches of the NLANR caching hierarchy and they all cooperate to share their load.

To analyze the impact of the NLANR caching hierarchy on the polling rate of a popular origin server, we choose the CNN server which is one of the most popular Web sites [6]. CNN Web server provides news at any time of the day with no fixed schedule. We have considered the CNN welcoming page (http://www.cnn.com/, http://cnn.com) and we determined the total polling rate for this page for different max-age values, with and without the NLANR caching hierarchy. To obtain the topology of the NLANR caching hierarchy, we identified the different caches at the institutional or intermediate levels connected to the NLANR top-level caches that make a request for the CNN page. To simulate different max-age values, we have supposed that the CNN server sets a max-age value which varies from zero to two hours.

In Figure 3.6(a), we see the trace-driven simulation results for the CNN server, which are very similar to those ones obtained with our analytical model (Figure 3.5). When max-age is equal to zero, all polls go to the origin server regardless of the existence of the NLANR caching hierarchy. However, when the NLANR caching hierarchy is in place and the origin server sets a max-age of few minutes, many if-modified-since requests are filtered at intermediate caches, cutting down on the number of polls to the CNN origin server by more than one order of magnitude.



(a) CNN. (b) Sportszone.

Figure 3.6: Trace-driven analysis. Poll rate at the origin server with and without NLANR top-level caches for different max-age values.

In Figure 3.6(b) we have also analyzed an automated pull for the case of a less popular document. For this purpose we considered 10 days of logs from the welcoming page of the Sportszone Microsoft channel (http://channel-espn.sportszone.com/). As already showed in our analysis (Figure 3.5), the benefits offered by a caching hierarchy are smaller for unpopular documents than for popular documents. However, even for less popular documents the savings of hierarchical caching are still considerable, i.e., for the Sportszone page the NLANR caching hierarchy offers a reduction on the number of polls to the origin server of a factor of 2 or more as soon as a max-age of few minutes is set (Figure 3.6(b)).

3.3.2 Disk Space Analysis

For automated pull through a caching infrastructure to have low bandwidth usage and few polls at the origin server, caches should keep a copy of all requested documents while they are up-to-date. In this section we develop analytical models to calculate the disk requirements of a cache at a certain cache tier. Let N be the total number of documents. We assume that documents change periodically every update period t, thus, documents are removed from the caches every t time units. Denote S the average size of a Web document. Let β be the request rate for the N documents. We order the documents so that document 1 is the most popular document and document N is the least popular. We suppose that the probability that a request is for the i-th document is given by the Zipf distribution [94] [19], i.e. $\frac{\eta}{i^\alpha}$, where α is the Zipf parameter and η is given by $\eta = (\sum_{i=1}^N i^{-\alpha})^{-1}$. We suppose that each document is requested independently from other documents. The request rate for the i-th document therefore is $\beta\frac{\eta}{i^\alpha}$. The parameter α varies between 0.64 and 0.8 [19], taking smaller values at the high level caches which receive requests already filtered by the lower level caches.

To quantify the disk requirements of the caches, we calculate the disk space D, i.e. the disk space needed to keep a copy of every requested document while the document is up-to-date. Since documents expire after t time units, a cache needs to keep all documents that have at least one request during a period t. If caches do not have sufficient storage capacity, some documents will be evicted from the cache before they are expired and thus, the performance of a pull distribution with caching infrastructure will decrease. The required disk size for a cache is given by

$$D = S \cdot \sum_{i=1}^{N}(1 - e^{-\beta\frac{\eta}{i^\alpha}t}),$$

since $1 - e^{-\beta\frac{\eta}{i^\alpha}t}$ is the probability that document i has at least one request during a period t.

Table 3.1 presents the required disk space for different request rates β, different Zipf parameters α, and different update periods t. We suppose $N = 1$ million Web documents require an automated delivery. We have chosen values of α and β that could correspond

Table 3.1: Disk space requirements at different levels of a caching hierarchy. $S = 10$ KB.

β, α	5 days	10 days	15 days
β= 1 req/s α=0.8	854 MB	1.4 GB	2.1 GB
β= 10 req/s α=0.64	3.5 GB	5.4 GB	6.4 GB

to those of an institutional and top-level cache [79] [6]. As the period t increases, the disk requirements also increase since the probability that a document is requested in a period t

increases. In the limit, for an infinite update period t and a large enough client population, the cache would need a disk space equal to $N * S$ KBytes. From Table 3.1 we see that for a request rate equal to 1 req/sec and for an average update period equal to 15 days, the disk space required at an institutional cache is equal to 2 GBytes. As the request rate increases, there are more documents that receive at least one requested during a period t, and the disk space required increases as well. For a top-level cache with a request rate equal to 10 req/sec the disk space required is about 6 Gbytes. These analytical values are very close to those ones reported in different trace-driven simulations and real caches with similar parameters β, and α [19] [45] [35].

Of course, a caching infrastructure requires cache servers to be purchased – perhaps a large number for each ISP in order to have sufficient storage, processing power, and access bandwidth. But ISPs are prepared to pay the cost, as they are currently aggressively deploying caches in their networks [12]. In fact, it is quite common to find institutional caches with capacities in the order of GB and top-level caches in the order of several tens of GB [6] [80] since the price of the disks is dropping very fast [20].

3.4 Pushing with Caching Infrastructure

As we have discussed in the previous section, in the cases that i) documents are updated at fixed intervals, or ii) documents are updated randomly and origin servers set a small max-age interval, pull through hierarchical caching uses bandwidth efficiently and significantly limits update checks sent to the origin server as well as document transfers. However, when documents are updated *randomly* and origin servers can not set even a small time interval during which documents remain unchanged, automated pull generates many requests to the origin server and does not scale. In this latter case, a push distribution scheme should be implemented.

If unicast is used to push document updates from the origin server to a large number of clients, two main problems arise: i) the origin server needs to keep a list with all the interested clients, ii) bandwidth is wasted because the same document copy is transmitted multiple times through the same links. An alternative scheme is to multicast from the server to the clients. Using multicast, the server does not need to keep state information about the interested clients; the server only needs to remember the multicast address where to push document updates. Additionally, multicast uses bandwidth very efficiently. However, multicast requires all receivers to be synchronized to obtain a document update and in the case where many documents are delivered via multicast, it may lead to an unscalable number of multicast trees or to non-trivial decisions on how to group different documents to use the same multicast tree. In addition to these considerations, multicast is not yet widely available as an end-to-end service in the public Internet [44], and there are still some technical issues that are not totally resolved, e.g, congestion control, scalable intra-domain multicast [55]. An alternative solution to IP-multicast, is to use a caching infrastructure. A caching infrastructure is already a fact in much of the Internet and it can easily perform application-level filtering, forwarding every document update to those areas where there are interested receivers, and providing strong consistency.

A *push with caching infrastructure* scheme can work as follows. Clients subscribe to a document and the subscription is propagated up in the caching hierarchy to the origin server. All intermediate caches in the path from the client to the origin server also get subscribed to the document. A subscribed cache at level l, records the document's URL and the addresses of those caches at level $l-1$ that are also subscribed to the document. Origin servers keep the addresses of the top-level caches that are subscribed to their documents. When origin servers update a document, they push the document update to the subscribed top-level caches using unicast and an extension of the HTTP protocol [24]. Top-level caches that receive a document update but that do not have subscribed children caches do not forward the update to the next tier. Top-level caches with subscribed children caches push the document update via unicast to the subscribed caches in the next tier. This process is repeated at every level of the caching hierarchy, until the document update gets to the subscribed institutional caches. As an alternative to pushing the document update, origin servers/caches could send a small *notification message* to the subscribed children caches at the next tier using an extension of the ICP protocol [61] [37]. This notification message would inform children caches about the new document update, and then, children caches would pull the updated document from the origin server/parent caches with the existing HTTP protocol.

At the institutional level, institutional caches could also keep state information about subscribed clients and push the document update to them. (However, since clients may be disconnected during some periods of time, pull operation may be better suited for the clients). Subscriptions in the server/caches expire after a certain *lease time* and would need to be periodically refreshed by the clients [58] [92]. A hierarchical caching push also requires mechanisms for establishing and maintaining the caching hierarchy, as discussed in [93]. Intermediate caches can purge the document update once it has already been pushed to the institutional caches, thus, reducing disk space requirements. However, intermediate caches could also keep the document updates to satisfy requests from clients that are not subscribed to this document but still want to hit the document at a nearby cache.

3.4.1 State Information at the Caches

With a hierarchical caching push, intermediate caches do not necessarily need to keep copies of the documents; however, for every document they need to keep state information about which caches at the next tier are subscribed. We now calculate the per-document state information at the origin server and the intermediate caches without and with a caching hierarchy [1]. Let $Q(p)$ be the expected number of entries for one document at the origin server when there is no caching hierarchy, where p is the probability that an institutional ISP has a client interested in a document. We have

$$Q(p) = O^{LH} p \ . \tag{3.3}$$

In the case of a caching hierarchy, the origin server keeps a list of the subscribed top-level caches. Top-level caches only keep a list of the subscribed caches at the lower level, and so on. Note that $1 - (1-p)^{O^{(l-H)}}$ is the probability that a cache at level $l - H$ is subscribed to a document. Let $Q_l^h(p)$ be the number of entries at a server/cache at level l,

[1]Recall that institutional caches do not need to keep state information since they do not have any children caches

$l \in \{H, 2H, ..., LH\}$, and $Q^h(p)$ the average number of entries in a server/cache. We have

$$Q_l^h(p) = O^H \cdot (1 - (1-p)^{O^{(l-H)}}) \tag{3.4}$$

$$Q^h(p) = \sum_{l \in \{H,2H,...,LH\}} \frac{O^{LH-l}}{(O^{LH}-1)/(O^H-1)} Q_l^h(p) \ , \tag{3.5}$$

because O^{LH-l} is the number of server/caches at level l, and $\sum_{i=0}^{(L-1)H} O^i = \frac{O^{LH}-1}{O^{H-1}}$ is the total number of server/caches excluding the institutional caches. In Figure 3.7 we plot the number of entries per-document in the origin server and in the caches for a caching hierarchy and no caching hierarchy as a function of p. We see that a caching hierarchy reduces the average

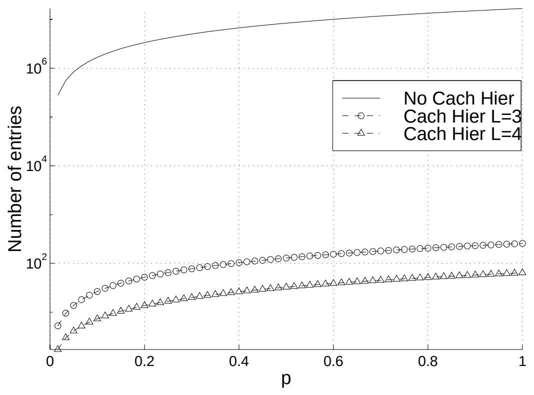


Figure 3.7: Number of entries at the server for no caching hierarchy and average number of entries at the server/caches for a caching hierarchy. $O = 4$, $LH = 12$ links.

number of entries in a server/cache by four orders of magnitude compared to the case where there is no caching hierarchy. Additionally we see that the higher the number of tiers L in the caching hierarchy, the lower the average number of entries in the server/caches since every server/caches needs to keep track of fewer children caches. In Figure 3.7 we see that it is enough to have a 3 level caching hierarchy to reduce the per-document state information by a factor of at least 10^4.

Even though the per-document state information with a caching hierarchy is not very high, caches need to keep state information for *every* document that uses a pure-push distribution. If we assume that there are 1 million documents that require an automated delivery, and that for every page 58 bytes of state information are required (which is enough to store a URL, and keep track of 64 children-caches), top-level caches need to keep at most 55 MB of state information, which is a reasonable value given the current storage capacities of Web caches [79]. However, from the total number of documents that require an automated delivery only few of them must be delivered with a pure-push distribution and can not be delivered with an automated pull and a caching infrastructure (see Section 3.3). In addition, assigning a certain time-to-live to every subscription, i.e. a lease time [58] [92], caches can reduce even more the number of documents for which they need to keep state information, since only popular documents will frequently refresh their subscription. Also, when there are multiple top-level caches, every top-level cache can take care of a different portion of the documents, thereby, sharing the load. Documents can be group into categories (i.e., CNN sports) and caches only need to keep an entry for every category and not for every document. Top-level caches could charge origin servers that want to push their latest information into the caching infrastructure, discouraging origin servers from using a hierarchical cache push distribution unless really necessary. Finally, in next section we present a push distribution for popular documents which combines caching and multicast, and avoids all the state information at the caches for popular documents.

3.4.2 Bandwidth Analysis

We now determine simple expressions for the *bandwidth usage* to push a document as a function of the probability p that an ISP has a client interested in a document for i) no caching infrastructure (unicast), ii) caching infrastructure, and iii) IP-multicast. Based on these simple expressions, we study how many cache tiers are needed for a caching infrastructure to mimic multicast in terms of bandwidth. We define bandwidth usage as the total number of links traversed to distribute a document update from the origin server to each institutional ISP with interested clients.

We assume that the total demand for a document is uniformly distributed among all institutional ISPs (at the end of this section we also consider random network topologies and non-uniform document demand). Using a uniformly distributed document demand we try not to be biased in favor of multicast or hierarchical caching; if the demand were concentrated around some nodes, the benefits of multicast or hierarchical caching would be much higher due to the high sharing.

In the case of **unicast** the expected bandwidth usage, $BW_u(p)$, is the number of links between the origin server and the institutional ISP, LH, times the expected number of institutional ISPs with interested clients, $O^{LH}p$, i.e.,

$$BW_u(p) = LHO^{LH}p \ . \tag{3.6}$$

For the case when all institutional ISPs require the document update, $p = 1$, the bandwidth needed is $BW_u(1) = LHO^{LH}$.

The expected bandwidth usage with a caching hierarchy, $BW_h(p)$, is the number of links H between every cache level times the expected number of caches at a given level that require the document update. A cache at level l requires the document update with a probability $1 - (1 - p)^{O^l}$, which is the probability that at least one of the O^l institutional caches rooted at level l is interested in that document. Thus,

$$BW_h(p) = \sum_{l \in \{0, H, 2H, \ldots, (L-1)H\}} O^{LH-l} H(1 - (1 - p)^{O^l}). \tag{3.7}$$

When all institutional caches are interested in a document $BW_h(1) = H\frac{O^{(L+1)H} - O^H}{O^H - 1} \simeq HO^{LH}$, for $O^H >> 1$.

In the case of **IP-multicast**, packets can be replicated at every network level and share all network links from the origin server to the institutional ISPs. The expected bandwidth usage $BW_m(p)$ is the sum of the expected number of links traversed at every network level. The number of links traversed between level $l + 1$ and level l is equal to $O^{LH-l}(1 - (1 - p)^{O^l})$, where $1 - (1 - p)^{O^l}$ is the probability that the multicast tree is extended to level l. Thus,

$$BW_m(p) = \sum_{l=0}^{l=LH-1} O^{LH-l}(1 - (1 - p)^{O^l}) \ . \tag{3.8}$$

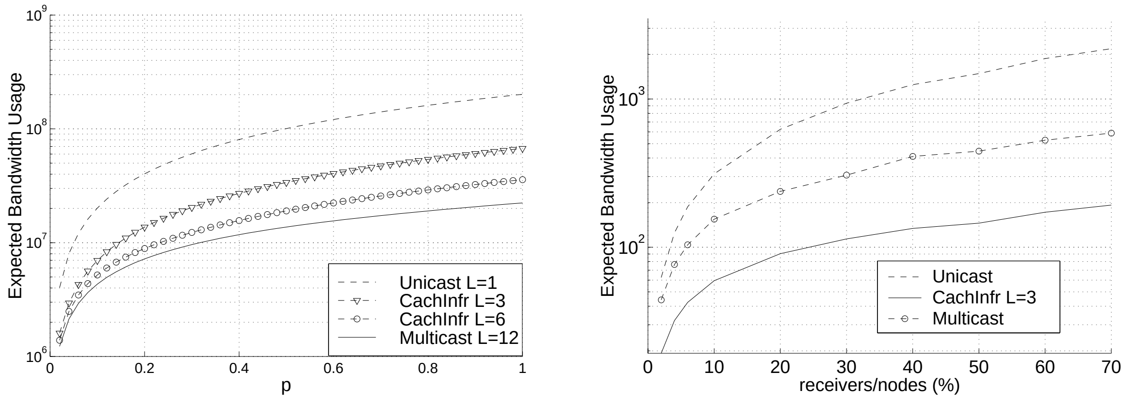
When all institutional ISPs require the document update, $p = 1$, we have $BW_m(1) = \frac{O^{LH+1} - O}{O - 1} \simeq O^{LH}$, for $O >> 1$.

Table 3.2 summarizes the bandwidth usage of unicast, multicast, and caching for $p = 1$ and $O >> 1$. We observe that a caching hierarchy reduces L times the bandwidth usage compared to unicast. Thus, using a caching hierarchy is attractive when there are many cache tiers L. Multicast uses H times less bandwidth than hierarchical caching and LH times less bandwidth than unicast. Thus, when there are few cache tiers and the height of each ISP is large, multicast uses significantly less bandwidth than caching, however, when there is a large number of cache tiers hierarchical caching uses similar bandwidth than multicast.

Table 3.2: Bandwidth usage for unicast, caching and multicast distribution. $p = 1$, $O >> 1$.

	Unicast	Caching Hierarchy	Multicast
Bandwidth	LHO^{LH}	HO^{LH}	O^{LH}

In Figure 3.8 we plot the bandwidth usage of the various schemes as p varies. For $L = 1$



(a) Full O-ary tree and uniform document demand. Total height of the tree $LH = 12$ links. $O = 4$.

(b) Random network topology and non-uniform document demand.

Figure 3.8: Bandwidth usage for unicast, multicast and hierarchical caching.

there is no caching hierarchy and there are only caches at the institutional level. In this situation, the server and the institutional ISPs communicate via unicast. Comparing unicast and multicast, we see that the difference in bandwidth usage is rather pronounced once a document becomes moderately popular. For very popular documents, $p = 1$, unicast uses approximately $LH = 12$ times more bandwidth than multicast. For a caching infrastructure we have varied the number of cache tiers L from 3 to 6. As the number of tiers L increases, the bandwidth usage of a caching hierarchy gets closer to the bandwidth usage of multicast. Given that the total height of the tree $LH = 12$ links, a caching hierarchy with 12 cache tiers ($L = 12, H = 1$) perfectly mimics a multicast distribution in terms of bandwidth, however, this would mean that there is a cache at every multicast router. For a reasonable number of cache tiers (i.e., $L = 4$), a caching infrastructure considerably reduces the bandwidth usage compared to no caching infrastructure, and has a bandwidth usage close to that of multicast, since it performs an application-level multicast distribution.

In Figure 3.8(b) we considered the case of a random network topology and non-uniform document demand. To simulate this scenario we used *tiers* [33], which is a random topology generator. We created a hierarchical topology of three levels: WAN, MAN, and LAN that aim to model the structure of the Internet topology. A WAN connects 20 MANs, and each MAN connects 9 LANs, resulting in a core topology of 225 nodes. To simulate a three level caching hierarchy we placed Web caches at the border nodes between the LANs and MANs, and the MANs and the WAN. To simulate non-uniform demand we place receivers at random locations. We defined the bandwidth usage as the number of links traversed to deliver one packet from the source to the receivers. We calculated the bandwidth used by unicast, multicast with shortest path tree, and hierarchical caching. From Figure 3.8(b) we observe that the results obtained with the random topology generator are very similar to those obtained with our analytical model (Figure 3.8(a)). We also see that a simple three level caching hierarchy considerably reduces the bandwidth usage compared to a unicast distribution.

3.5 Pushing with Multicast and Caching Infrastructure

Pushing popular documents through a caching hierarchy uses more bandwidth than IP-multicast because communication among caching levels uses unicast. Using IP-multicast to push all the popular documents uses bandwidth very efficiently but requires a large number of multicast trees (one per document of group of documents) and an end-to-end IP-multicast infrastructure. Next, we combine hierarchical caching push (Section 3.4) with multicast among cache tiers to push to a large number of popular documents using few multicast trees, and requiring only local (not end-to-end) IP-multicast.

To push popular documents among cache tiers using multicast, origin servers/caches are connected to children caches in the next tier with a unique multicast tree. Servers/parent-caches dynamically decide on a per-document basis whether to multicast a document to all children caches in the next tier or to unicast the document only to those subscribed children caches. The decision of multicasting or unicasting among cache tiers is based on document's popularity and tries to minimize bandwidth usage. If the document is very popular, there will be many subscribed children caches in the next tier and thus multicast to *all* children caches will use lower bandwidth than unicast to the *subscribed* children caches. Hierarchical caching push with multicast only needs a *fixed and small* number of *static* multicast groups, independent of the number of documents, and equal to the number of server/caches in the L top tiers, $\frac{O^{LH}-1}{O^{H}-1}$. Using hierarchical caching push with multicast alleviates the load on the multicast routers because the number of multicast groups and, therefore, the number of multicast entries is smaller and constant. When multicast is used to push popular documents among cache tiers, parent caches and servers do not need to keep a list of entries with the subscribed children caches. Instead, they only need to know the multicast address of the multicast tree where children caches are joined. In addition, servers/caches experience less load since they only need to send one copy of the popular documents towards the next-tier caches. With hierarchical caching push and multicast there is no need for end-to-end IP-multicast and it is enough to have intra-domain multicast [55] inside every ISP, which is easier to deploy.

Next, we determine the *document popularity threshold* for which multicast should be employed among cache-tiers instead of unicast. Multicast shares all common paths inside an ISP and has the same bandwidth usage independently of how many children caches are subscribed. On the other hand, the bandwidth usage of unicast increases linearly with the number of subscribed children caches. When the bandwidth used by multicast from tier L to tier $L-1$,$\sum_{i=1}^{H} O^i$, is lower than the bandwidth of unicast from tier L to the subscribed caches at tier $L-1$, $HO^H(1-(1-p)^{O^{l-H}})$, we propose that servers/parent-caches multicast the document update to all its children caches. The bandwidth used by a cache tier-to-tier multicast distribution BW_b at every network level is the minimum of the bandwidth usage of multicast and unicast. Note that O^{LH-l} is the number of server/caches at cache tier l. BW_b is given by

$$BW_b(p) = \sum_{l\in\{0,H,2H,\ldots,(L-1)H\}} O^{LH-l} \cdot min\{\sum_{i=1}^{H} O^i, HO^H(1-(1-p)^{O^{l-H}})\} \ , \qquad (3.9)$$

For very popular documents, $p \to 1$, $\sum_{i=1}^{H} O^i < HO^H$, and the bandwidth used by a tier-to-tier multicast distribution $BW_b(1)$ is equal to the bandwidth used by multicast $BW_b(1) = \frac{O^{(LH+1)}-O}{O^H-1} = BW_m(1)$. In Figure 3.9 we plot the number of links used by hierar-

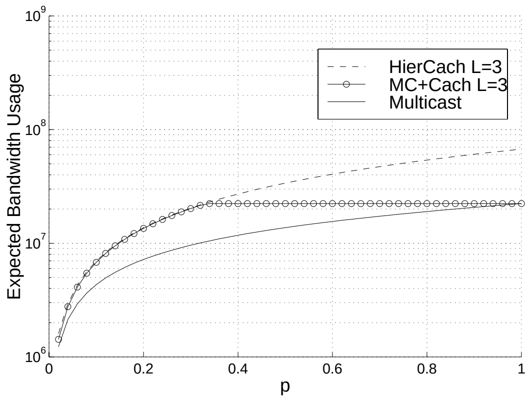


Figure 3.9: Number of links used by hierarchical caching push with unicast among cache tiers, an IP-multicast distribution, and a cache tier-to-tier multicast distribution. Total height of the tree $LH = 12$ links. $O = 4$.

chical caching push with a unicast communication among cache tiers, by IP-multicast, and by a hierarchical caching push with multicast communication among cache tiers depending on how many institutional caches are subscribed to a document. We see that for unpopular documents, a cache tier-to-tier multicast distribution uses as much bandwidth as hierarchical caching push with unicast. After a certain popularity threshold ($p = 0.33$), servers/parent caches with enough subscribed children caches start distributing the document update via multicast among cache tiers, using less bandwidth than hierarchical caching push with unicast. For very popular documents ($p \to 1$), a tier-to-tier multicast distribution uses the same bandwidth than a multicast distribution while reducing the complexity of managing multiple dynamic multicast trees.

It is important to notice that in a real network topology the switching point between unicast or multicast would not only depend on the documents' popularity but also on the network location of the destination hosts, since the multicast cost depends on the network topology. Thus, to switch among unicast or multicast, parent caches need to take into account the network position of the subscribed caches at the next tier and calculate how much sharing exists among them, e.g., if two subscribed caches are very close, the number of links shared among them is very high and parent caches would switch from unicast to multicast even for less popular documents since the cost of multicast is very small.

3.6 Conclusions

In this chapter we have studied the impact of a caching hierarchy to reduce the number of if-modified-since polls to the origin server and proposed new ways in which a caching hierarchy can be used to push information in a scalable way. First, we have showed that a caching hierarchy can efficiently cut down the number of if-modified-since polls to the origin server as soon as the servers can set a small time interval (i.e., few minutes) during which documents do not change.

In the situation where servers can not specify even a small time interval during which documents remain unchanged, automated pull generates too many polls at the origin server. For this later case, we have proposed a new scheme that pushes documents through a caching hierarchy as soon as a document changes. Using push with caching infrastructure, caches act as application-level routers and filters, forwarding the document only to those caches with subscribed clients. Push with caching infrastructure requires little state information at the server and caches and minor modifications to the existing protocols. Also it uses bandwidth very efficiently, approaching the performance of multicast when there are few cache tiers (i.e. three to four). Finally, we have considered push with caching infrastructure with multicast among cache tiers to reduce the number of multicast trees needed to distribute a large number of documents. We showed that combining a caching infrastructure with local multicast among cache tiers, can use bandwidth as efficiently as an IP-multicast distribution and keep the number of multicast trees fixed and independent of the number of documents. Additionally, to implement cache tier-to-tier multicast it is sufficient to have local *intra-domain* multicast at every ISP and global Internet-wide multicast is not required.

Chapter 4

Large Caches and Satellite Distribution

In this chapter we discuss the performance of a document distribution model that interconnects Web caches through a satellite channel. During recent years Web caching has emerged as an important way to reduce client-perceived latency and network resource requirements in the Internet. Also a satellite distribution is being rapidly deployed to offer Internet services while avoiding highly congested terrestrial links. When Web caches are interconnected through a satellite distribution, caches end up containing all documents requested by a huge community of clients. Having a large community of clients connected to a cache, the probability that a client is the first one to request a document is very small, and the number of requests that are hit in the cache increases. In this chapter we develop analytical models to study the performance of a cache-satellite distribution. We derive simple expressions for the hit rate of the caches, the bandwidth in the satellite channel, the latency experienced by the clients, and the required capacity of the caches. Additionally, we use trace driven simulations to validate our model and evaluate the performance of a real cache-satellite distribution.

4.1 Introduction

The growth of the World Wide Web overloads popular servers, increases the network traffic, and causes slow responses to the clients. To alleviate these problems, Web caching is being extensively deployed in the Internet. With Web caching when a client requests a document for the first time, the document is delivered directly from the origin server and a copy of the document is stored in the cache. Further requests for the same document are satisfied directly from the cache. Web caching happens at different network levels: Clients have local caches that satisfy multiple requests for the same document coming from the same client (i.e. temporal locality). Local ISPs have institutional caches that satisfy requests for the same document coming from different clients (i.e., geographical locality). Requests satisfied by a cache are called *hits*. Requests not satisfied by a cache are called *misses*. misses can be classified into:

- *First-Access*: misses occurring when requesting documents for the first time.
- *Capacity*: misses occurring when accessing documents previously requested but discarded from the cache due to space limitations.
- *Updates*: misses occurring when accessing documents previously requested that have expired in the meantime.
- *Non-cacheable*: misses occurring when accessing documents that need to be delivered from the origin server (e.g. dynamic documents generated from cgi-bin scripts or fast changing documents)

Even when a cache has infinite storage capacity, the number of misses in an institutional cache can be very high (50%-70%) [6] [12]. Recent studies have shown that non-cacheable documents typically account for 10%-20% of all requests[85][65]. Furthermore, update misses account for approximately 9% of all requests [85]. Thus, there is a large percentage of requests that result in first-access misses (30%-50%), in particular when a small client population is connected to the cache.

One way to reduce the number of both first-access, and update misses is to preload the cache with new documents and document updates, expecting that clients are likely to request them later. Prepopulating caches, however, requires additional disk space and may waste network bandwidth. Disk space may not be such a problem since disk capacity is increasing at rate of 60% per year and large disks are becoming increasingly cheaper [20][46]. A more serious problem arises when network bandwidth is wasted by prefetching documents that no client requests through highly congested and expensive Internet links. An alternative way to preload Web caches is to use a satellite distribution, which has fewer losses and congestion problems than a distribution in the terrestrial Internet. Also, a satellite distribution can reach a very large number of receivers with low cost and relatively little effort- adding a new additional receiver does not increase the cost of transmission.

A *cache-satellite* distribution works as follows. Clients are connected to caches that have a satellite receiving dish. When there is a miss at any cache, the cache obtains the document from the origin server and reports the missed URL to a *master* site that is equipped with a satellite transmitting antenna. The master site fetches itself the document from the origin server and transmits the document over the satellite channel. As a result, *all* caches connected to the cache-satellite distribution receive the broadcasted document and can use it to satisfy local requests. The probability that a cache is the first cache asking for a document becomes very small, thus, reducing significantly the number of first-access and update misses. As more caches join the satellite distribution, the aggregation of clients is higher and the number of misses in one cache is smaller. After a certain period of time, caches with huge storage capacity can end up containing most of the documents in the Web. Recently, several companies (e.g., SkyCache [83]) have started to offer such a cache-satellite distribution service. For a more detailed description of the technical details of a cache-satellite distribution see [83][78].

In this chapter we investigate the feasibility and performance of a cache-satellite scheme. In the first part of the chapter we develop analytical models to calculate upper bounds on the performance of a cache-satellite distribution. Using an analytical model, we can study the effect of many different parameters over a large range of values and examine the implications of future trends in Web traffic. In particular, we derive expressions to calculate the maximum achievable hit rate in a cache connected to a cache-satellite satellite distribution and study how the hit rate varies with the client population. We also analyze the latency experienced by clients when caches cooperate through a satellite distribution and when caches cooperate through a caching hierarchy, which is commonly used in the Internet to make caches cooperate [23]. In addition, we calculate the storage capacity needed to store all documents pushed through the satellite as well as the bandwidth requirements for the satellite link. Exploiting the highly skewed distribution of Web documents, where very few documents account for most of the requests, we also study the performance of a satellite distribution where only those documents that see a certain number of requests during the lifetime of the document are pushed through the satellite link. We study the impact of this simple filtering policy on the hit rate, bandwidth, and disk space.

In the second part of the chapter we use trace driven simulations to evaluate a real cache-satellite distribution. In particular we examine logs from an ISP in the USA (AYE) that is connected to the SkyCache satellite distribution [83]. This cache-satellite distribution covers the USA and Europe and comprises a large number of ISPs. We study the hit rate improvement offered by a cache-satellite distribution and the additional disk space requirements. We then use NLANR [6] traces to explore the potential of a cache-satellite distribution with a higher number of clients. For this purpose, we analyze traces from AYE and NLANR that were collected during the same period of time and simulate a scenario where NLANR caches also get connected to the cache-satellite distribution.

Our results show that a cache-satellite distribution has important benefits for ISPs with a small client population. ISPs with few clients can achieve very high hit rates *locally*, without having to install and maintain any cache-cooperation protocol for creating a larger receiver population. A cache-satellite distribution, automatically federates a large number of ISP caches with minimum configuration and very little effort, increasing the hit ratios and providing fast access to a large number of documents. In the case of an ISP with a large client population, connecting itself to a cache-satellite distribution does not increase the hit rate significantly. However, a cache-satellite distribution can significantly reduce the cost of any ISP to fill the cache with many documents, since documents reach the cache through the satellite link, allowing the more expensive terrestrial links to be used for other services.

The rest of the chapter is organized as follows. Section 2 discusses related work and similar approaches to increase the performance of Web caches. In Section 3 we present the model of a cache-satellite distribution that will be used for our analysis. Section 4 derives expressions to evaluate the performance of a cache-satellite distribution. In Section 5 we explore the impact of several model parameters on latency, hit rate, disk capacity, and bandwidth. Section 6 uses trace driven simulation to study the performance of a cache-satellite distribution. The results obtained in Section 6 are also used to validate the model developed in Section 3. The chapter ends with a conclusion and summary of the results.

4.2 Related Work

Web caching has been recognized as one of the most important techniques to help scaling the Internet. The hit rate of a Web cache is a function of the client population it manages. During the last years there has been extensive research on how to make caches cooperate to increase the total effective client population, increase the hit ratios, and reduce the document-access latency. Web caching cooperation was first proposed in the context of the Harvest project [23], that designed the Internet Cache Protocol (ICP) [89], which supports discovery and retrieval of documents from neighboring caches. Today, many caches have established hierarchies of caches that cooperate via ICP [6]. Other approaches to make caches cooperate have been proposed recently, such as the Cache Array Routing Protocol (CARP) [86], the central directory approach (CRISP) [43], Summary Cache [37], Cache Digest [80], the Relais project [64]. All these approaches use different strategies to share meta-information indicating the location of Web documents in Web caches.

Cache cooperation needs a careful selection of the cooperating caches and in some cases a whole new application-layer routing infrastructure with intermediate caches in the path from the local ISP to the origin server [87] [67]. Additionally, requests not hit in the local ISP need to travel to other ISP caches, which may be overloaded or connected through congested links. If many documents could be prefetched into local caches, most of the requests could be satisfied locally and no intermediate caches or cooperating protocols would be needed. There have been several approaches to preload documents into Web caches [38] [59] [25]. However, all these approaches use the terrestrial Internet links and incur a high cost if the prefetched documents are not requested. In this chapter, on the other hand, we present a scheme where documents requested by a large community are pushed into local caches using a satellite distribution, which is cost-efficient and does not require any inter-cache cooperation.

4.3 The Model

For the network topology and for hierarchical caching we use the model described in Section 2.2.

4.3.1 Satellite Distribution

Given the model of the Internet described in Section 2.2, we now consider the situation where the O^{2H} institutional caches cooperate via a satellite distribution (Figure 4.1). Every institutional ISP has an institutional cache with Internet connection, high storage capacity, and a satellite dish for receiving. In addition to the institutional caches, there is also a master site which has an Internet connection and a satellite transmitter. Note that compared to Figure 2.4 there is no more a caching hierarchy (i.e. regional or national caches).

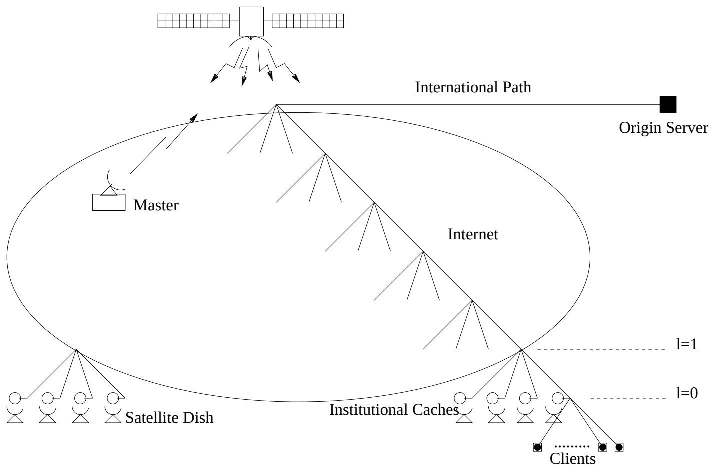


Figure 4.1: Cache-satellite distribution.

4.3.2 Web Documents

To study the implications of future trends in Web traffic and the impact of having many Web documents with different document's popularity, we use a more refined model for Web documents than the one presented in Section 2.2. We denote the total number of documents in the WWW in the year j-th as N_j. Let x be the annual rate at which the number of documents in the WWW is increasing.

$$N_j = N_{j-1} \cdot x.$$

Denote S_i, the size in bytes for Web document i, $1 \leq i \leq N_j$. Web document i expires after an elapsed update interval t following an exponential distribution with average update interval Δ_i [34], $f(t) = \frac{1}{\Delta_i} e^{-t/\Delta_i}$.

Let $\lambda_{I,i}$ be the mean request rate from an institutional ISP for document i, which is Poisson distributed [45]. Assuming that requests for document i are uniformly distributed among all O^{2H} institutional caches, there are $\lambda_{I,i} \cdot O^{2H}$ total requests for document i. Let β_I be the request rate from an institutional ISP for all N_j documents, $\beta_I = \sum_{i=1}^{N_j} \lambda_{I,i}$. β_I is Zipf distributed [19] [94] (see Section 3.3.2).

We consider that newly appearing documents will also be Zipf distributed. That is, there will be some new appearing documents that will be very popular but there will also be many other new documents that will be requested by few clients. Let θ be the percentage of requests that are for non-cacheable documents (private documents, cgi-bin, etc).

4.4 Performance Analysis

In this section we present analytical models to calculate upper bounds on the performance of a cache-satellite distribution. We derive expressions to calculate the achievable hit rate at an institutional cache connected to the satellite distribution and at an institutional cache not connected to the satellite distribution. We also derive simple expressions to calculate the bandwidth needed for the satellite link, the disk space requirements at the caches, and the document-access latency. The goal of this section is not to exactly model empirical results, but to examine the impact and sensitivity of a large range of parameters in a cache-satellite distribution.

4.4.1 Hit Rate

In Section 2.2 we calculated the probability to find an up-to-date document in any cache level of a caching hierarchy. In this section we develop simple analytical expressions to calculate the hit rate for *all* documents at several cache levels. The hit rate is the percentage of requests that find an up-to-date version of the requested document in the cache. We assume that the caches have an infinite capacity and therefore documents are not removed from the cache due to storage limitations. First, we analyze the hit rate at an institutional cache not connected to the satellite distribution and then we consider the hit rate at an institutional cache connected to the satellite distribution.

Let L_i be the network level where a document i is hit ($L_i = 0$ for the institutional cache). The steady-state hit rate Hit_j for all documents in an institutional cache in year j, is given by

$$Hit_j = \sum_{i=1}^{N_j} (\lambda_{I,i} \cdot P(L_i = 0)) \cdot \frac{(1-\theta)}{\beta_I} \tag{4.1}$$

where $P(L_i = 0)$ is given by equation 2.8.

When all institutional caches are connected to the satellite distribution, only the first request for document i out of all requests for the same document in all institutional ISPs sees a document miss. The rest of the requests will see a document hit. The hit rate at any institutional cache when O^{2H} institutional caches are connected to the satellite distribution can be calculated in the same way than for no cache-satellite distribution, using an effective client request rate equal to $\lambda_{I,i} \cdot O^{2H}$. Since the effective client population is larger, the probability that there has been already at least one request for document i before time τ when a client requests document i is given as $1 - e^{-\lambda_{I,i} O^{2H} \tau}$, and is close to one.

4.4.2 Disk Space

To quantify the disk requirements of the caches, we calculate the cache disk space D_j in year j. Caches need to keep a copy of every document distributed through the satellite during the period of time that the document is up-to-date. Using the derivation for the disk space obtained in Section 3.3.2, and the fact that the total number of requests for document i, $\lambda_{I,i} \cdot O^{2H}$, is equal to $\beta \frac{\eta}{i^\alpha}$, we can express the required disk space $D_j(t)$ in a cache for year j and a given update period t as

$$D_j(t) = S \cdot \sum_{i=1}^{N_j} \cdot (1 - e^{-\lambda_{I,i} O^{2H} t}), \tag{4.2}$$

where $(1 - e^{-\lambda_{I,i} O^{2H} t})$ is the probability that document i has at least one request during a period t. Thus, the average disk space D_j in year j is given by $D_j = \int_0^\infty D_j(t) \cdot f(t) dt$.

4.4.3 Bandwidth

Now we calculate the bandwidth needed for the satellite link to transmit all newly appearing documents as well as all document updates. The bandwidth BW_j needed in year j, is given by

$$BW_j = \sum_{i=1}^{N_j} bw_i \tag{4.3}$$

where bw_i is the bandwidth needed to transmit document i

$$bw_i = \int_0^\infty bw_i(t) \cdot f(t) dt \tag{4.4}$$

and $bw_i(t)$ is the bandwidth needed to transmit document i given an update period t. $bw_i(t)$ is given by

$$bw_i(t) = \frac{S_i}{t} \cdot (1 - e^{-\lambda_{I,i} O^{2H} t}). \tag{4.5}$$

4.4.4 Latency Analysis

Given the hierarchical topology of the Internet with very congested top levels and much less congested lower levels, documents found at low network levels close to the clients experience small latencies. To quantify the latency we calculate the expected number of network links that a request travels to hit a document. We analyze the latency in the case that institutional caches cooperate through a caching hierarchy and in the case that institutional caches are connected through a satellite distribution.

Let L_n be the number of links traversed by the $n-$th request for the same version of a certain document. The average number of links $E[L_n]$ traversed by the n-th request, is given by

$$E[L_n] = \sum_{l \in \{0,H,2H,2H+z\}} l \cdot P(L_n = l)$$

We now calculate the distribution of L_n. The first request $n = 1$ for a document in an update interval always travels to the origin server, i.e., $P(L_1 = 2H + z) = 1$. In the case that institutional caches cooperate through a caching hierarchy, for $n \geq 2$, $P(L_n = l)$ is given by

$$P(L_n = l) = \begin{cases} 1 - (1 - \frac{1}{O^{2H}})^{n-1} & l = 0 \\ (1 - \frac{1}{O^{2H}})^{n-1} - (1 - \frac{1}{O^{H}})^{n-1} & l = H \\ (1 - \frac{1}{O^{H}})^{n-1} & l = 2H + z \end{cases} \tag{4.6}$$

In the case that institutional caches are connected through a satellite distribution we have $P(L_n = 1) = 1$, for $n \geq 2$.

4.5 Numerical Results

In this section we pick some reasonable values for the different parameters used in the analytical models to obtain upper bounds on the requirements and performance of a cache satellite distribution. These values will be fix for the rest of the chapter unless stated differently.

The number of documents N_j in the Web at the beginning of the year j=1998 is about 250 millions [16]. The Web is increasing at a rate such that the number of documents get duplicated every year ($x = 2$) [16]. We consider that the HTTP traffic generated in all the caches connected to the satellite distribution ($O^{2H}\beta_I$) in $j = 1998$ is equal to $10,000$ document requests per second (this value is equal to the Internet backbone traffic in 1998 [20]). The HTTP traffic grows by a factor of 2.8 per year due to the increasing number of clients, the increasing period of time that clients are connected, and the increasing bandwidth of clients' connections [20].

We consider that there is no relationship between the update period of a document and its popularity [19]. Thus, we assume that all Web documents expire randomly with the same average update period $\Delta_i = \Delta$. We vary Δ from 1 hour to 20 days [34]. We also assume that the document size is independent of the document's popularity and the rate of change [19]. Therefore, we consider the same document size $S_i = S$ for every Web document, varying from 10 KB to 20 KB[57].

4.5.1 Latency Analysis

To consider real values for the latency, we analyzed 10 days of logs for the local cache at EURECOM, which is connected to a 3-level caching hierarchy. We averaged the latencies during the 10 days of the trace to obtain the following values: transmission time from the local cache= 117 msec, transmission time from the regional cache= 550 msec, transmission time from the national cache= 800 msec, transmission time from the origin server= 1183 msec.

Combining the probability $P(L_n = L)$ (equation 4.6) that the n-th request hits a document at level L, and the average latency to retrieve a document from network level L, we can calculate the average latency experienced by the n-th client request for a generic document.

In Figure 4.2 we considered the latency experienced by a client in a three level caching hierarchy and in a cache satellite distribution. In both schemes, we see that the first request for a document always needs to travel to the origin server, thus experiencing high latencies ($1,183$ msec). In hierarchical caching, the probability to find a document at closer to the clients increases slowly as the document becomes more and more popular. Thus, it is necessary to have a very high number of requests per update period ($n > 10^4$) for clients

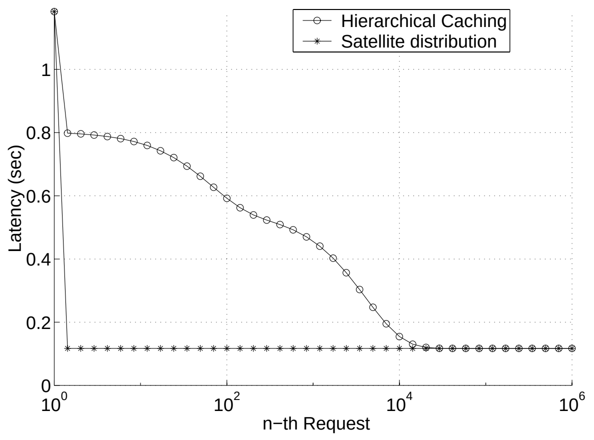
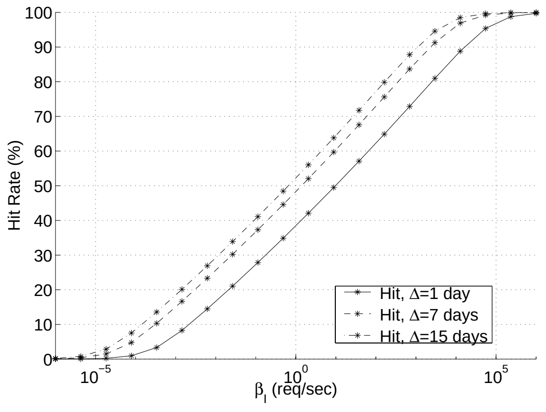


Figure 4.2: Average latency experienced by the n-th client in a 3-level hierarchical caching distribution and in a cache-satellite distribution. $O = 4$, $H = 3$, $z = 1$.

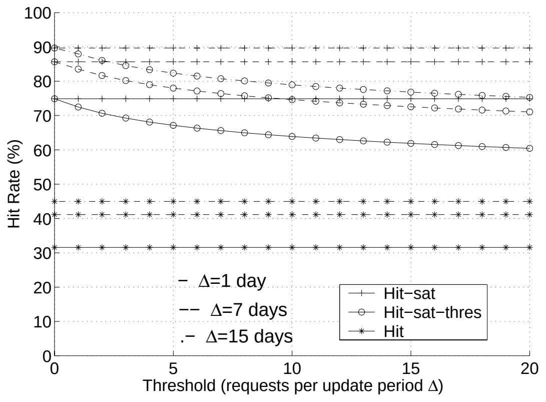
to experience average latencies equal to transmission time from the institutional cache. In a cache-satellite distribution, on the other hand, the document is pushed to all institutional caches after the first client request. Thus, every client request after the first one ($n > 1$), is directly satisfied from the institutional caches. As a result, a cache satellite distribution clearly reduces the latency to the clients by bringing documents into edge caches after the first document request.

4.5.2 Hit Rate

Figure 4.3(a) shows the hit rate for cacheable documents as a function of the total request rate (β_I) at a single cache (the population size can be easily calculated by assuming a certain client request rate, e.g. 500 requests/day [4]). We observe that for caches with small request rates, the hit rate is very small since the sharing among clients is very limited. As the number of requests increases, the hit rate increases with the logarithm of the request rate. However, it is necessary for a cache to have at least 100 to 1000 requests per second to satisfy 80% to 90% of the cacheable requests. This request rate is equivalent to having a population size of about $17,300$ to $173,000$ clients connected to the cache, which is a very large value for many ISPs. As a result, ISPs with small client populations usually deploy inter-cache cooperation protocols with other ISPs at the same or different network levels to increase the effective client population and improve their hit ratios.



(a) Hit rate for a single cache as a function of the request rate. (Log scale on x-axis).



(b) Hit rate for $O^{2H} = 64$ caches connected to the satellite distribution, and for a single cache, as a function of the popularity threshold.

Figure 4.3: Hit rate analysis ($\%$ of cacheable requests).

A satellite distribution, on the other hand, can easily increase the effective client population connected to the cache with no inter-cache communication protocol. In Figure 4.3(b) we considered the case of a single cache with $\beta_I = 15$ requests per second, and the case of $O^{2H} = 64$ identical caches that cooperate using a cache-satellite distribution and generate 960 requests per second. Again we plot the hit rate for cacheable objects. Our analytical results for the hit rate on a single cache not connected to the satellite distribution indicates values ranging from 33% to 45%, which are very similar to the actual hit rates reported for many institutional caches [6]. For a cache connected to the satellite distribution we obtain hit rates from 80% to 90%. Thus, when individual caches get inter-connected through a cache-satellite distribution, the hit rate doubles.

Let R_i be the number of requests for document i before the document is updated. Given the long-tailed distribution of Web documents, there are many documents that are requested only once or less in an update period. Documents requested only once in an update period are not shared by several clients and should not be cached. Therefore, the master site can take the decision to broadcast only those documents that have a number of requests per update period Δ higher than a certain *threshold*. In Figure 4.3(b) we also show how the hit rate for cacheable documents varies as a function of the threshold. For small thresholds and a satellite distribution, the hit rate for cacheable documents is less than 100% since there are many documents that are requested only once during an update period. As the threshold value increases, the hit rate decreases, however, even for very high threshold values, e.g. 20 requests per update period, the hit rate is not decreased by more than 15%. As we will see in next section, this is a very important result since using threshold filtering policies allows caches to achieve high hit rates while reducing the disk requirements and bandwidth needed for the satellite link.

4.5.3 Storage Capacity

The price of disks is decreasing faster than the price of the network capacity. Additionally, the capacity of the clients' disks is increasing at a rate of 60% per year [20] with a baseline of 9 GBytes in 1998 (Figure 4.4). In a few years it will be easy to find disks with capacities close to TBytes at current prices [46].

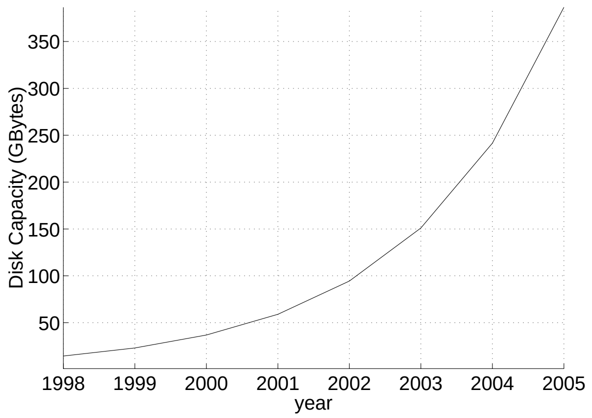
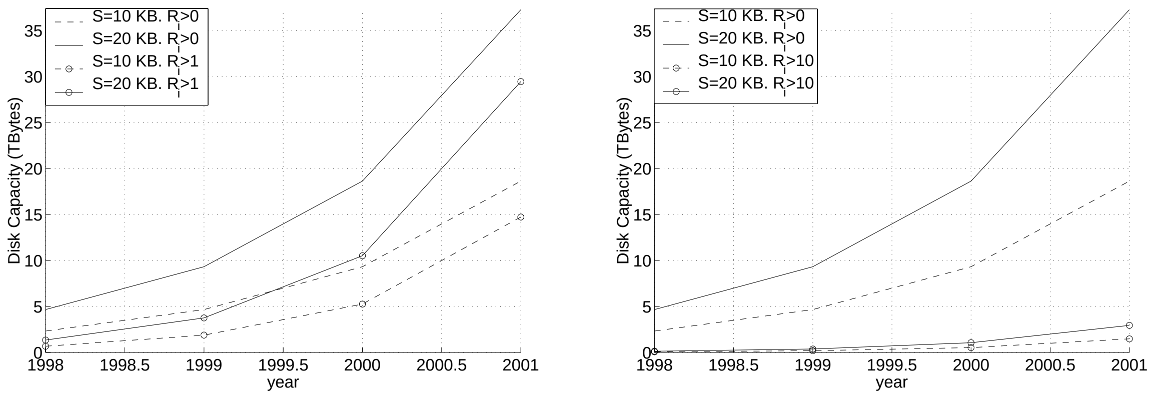


Figure 4.4: Disk Capacity Trend.

With such large storage capacities it will be feasible to store a large portion of the Web on several disks at different points in the network. Using equation 4.2, in Figure 4.5(a) we show the disk capacity needed to store all Web documents that have at least one request per update period ($R_i > 0$). The values presented are an an upper bound for the storage requirements in a cache. Taking an average document size of $S = 10$ Kbytes and $S = 20$ Kbytes, the total number of documents distributed through the satellite can be stored in several TBytes (i.e., 20 TBytes in the year 2000). These results match very well those reported in a recent study [57], where the size of the publicly indexable Web pages and images in the middle of 1999, was estimated to be 18 TBytes. The storage capacity to store all newly appearing Web documents needs to be doubled every year, following the rate at which the number of Web documents is increasing.



(a) More than once. (b) More than ten times.

Figure 4.5: Disk capacity needed to store all Web documents send through the satellite.

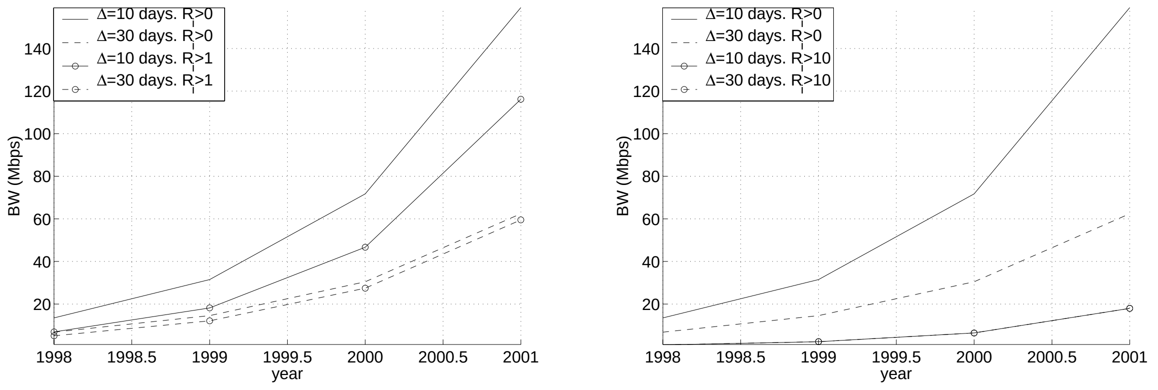
In Figure 4.5(a) we see that if the master side only transmits those documents requested more than once ($R_i > 1$), the needed storage capacity drops to 1.8 TBytes in year 1999 and increases at rate equal to the rate at which the HTTP Internet backbone traffic increases (2.8 times per year). In Figure 4.5(b) we see that if the master site only transmits those documents requested more than 10 times, the needed storage capacity in the year 1999 drops even more to 187 Gbytes and increases at a much smaller rate. Caches with storage capacities of several hundreds of Gbytes are already common today [79]. This simple filtering policy saves a lot of disk space in the institutional caches and reduces the hit rate at an institutional cache by less than 10% (see previous Section). In the case that caches do not have such storage capacities, more sophisticated filtering policies could be implemented at the client side to reduce the needed storage capacity while hardly affecting the hit rate [49].

4.5.4 Bandwidth

As we discussed in Section 4.4.3, the satellite needs to send document updates for existing documents and newly appearing documents in the Web. Using equation 4.3, we plot in Figure 4.6 the necessary bandwidth for the satellite link to keep broadcasting newly appearing documents and document updates that are requested at least once in an update interval ($R_i > 0$). We take the size of a Web document as $S = 10$ and vary the update period Δ from 10 to 30 days. From Figure 4.6(a) we can observe that the bandwidth needed to send documents through the satellite is close to 31 Mbps for year 1999 and that keeps increasing with a rate close to 2 per year. The rate at which the bandwidth increases depends on the rate at which the HTTP Internet backbone traffic increases and on the rate at which new documents appear in the Web. If all appearing documents were requested at least once in an update period, the bandwidth would increase at the same rate at which Web documents appear in the Web (factor of 2 per year).

As discussed in Section 4.5.3, there is a large group of documents that are only requested few times in an update interval. From Figure 4.6(a) we see that when the master site only transmits documents requested more than once ($R_i > 1$) the bandwidth needed is reduced to 19 Mbps for the case of $\Delta = 10$ days, and year 1999. When the update interval Δ increases, new updates must be distributed less frequently and the bandwidth needed decreases, i.e. for $R_i > 1$, $\Delta = 10$ days, and year 1999 the bandwidth needed for the satellite link drops to 12 Mbps. A satellite distribution of an analog television channel uses a bandwidth of 27 Mbps. Thus, with the capacity needed to broadcast one TV channel a cache-satellite distribution can be used to distribute almost all Web documents in the year 1999.

Figure 4.6(b) shows the bandwidth needed at the satellite link when the master site only transmits those documents requested more than 10 times. In this case, the needed bandwidth



(a) More than once. (b) More than ten times.

Figure 4.6: Required bandwidth at the satellite to transmit new documents and document updates. $S = 10$ KBytes.

at the satellite in 1999 is equal to 2.2 MBps. These numbers approximate those provided by the cache-satellite company Sky-Cache [83], which in 1998 was using a 4 Mbps link to feed Web documents into the subscribed caches and only broadcasted those documents that were requested several times during an update interval (e.g., more than two or three requests per update interval).

4.6 Trace-Driven Simulations

In this section we perform a trace-driven analysis to evaluate the hit rate of an institutional cache that it is not connected to a cache-satellite distribution and the hit rate of an institutional cache connected to a cache-satellite distribution. We also use the trace driven results to validate the analytical model developed in Section 4.4.

First, we will present the distribution of hits and misses in an institutional cache to better understand the potential benefits of a cache-satellite distribution.

4.6.1 General Distribution of Hits and Misses in a Single Cache

To investigate the distribution of hits and misses, we analyzed the logs of our institutional cache at Institut EURECOM [4]. We took 7 days of logs from the 10th of December to the 17th of December of 1998, which include $187,054$ requests. Our institutional cache has 6 GBytes of disk space and connects about 100 clients.

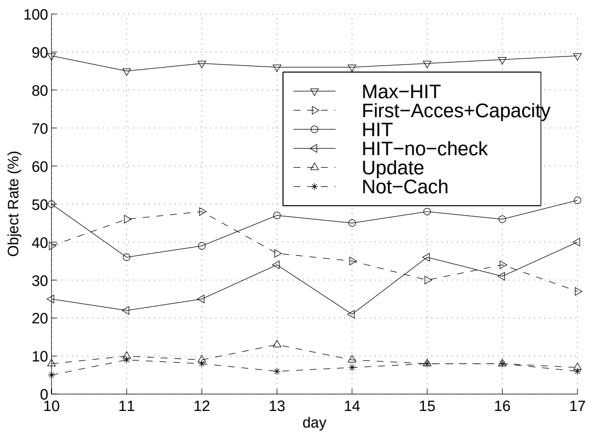


Figure 4.7: General distribution of EURECOM's cache entries. Hits are shown with a continuous line, misses are shown with a dashed line.

Figure 4.7 shows the distribution of the object rate for the different days of the logs (results for the byte distribution show a similar behavior). The total hit rate (*HIT*) includes documents that are directly satisfied from the cache and documents that are satisfied from the cache but need a previous check with the origin server. We see that the total *HIT* ranges from 35% to 50%. We see that the hit rate for documents for which the cache does not need to contact the origin server (*HIT-no-check*) ranges from 20% to 35%. The percentage of non-cacheable objects (*Not-Cach*) never exceeds 10%. We consider a document to be non-cacheable if 1) the document was created by a cgi-bin script, 2) the document requires authorization, 3) the document has an HTTP 1.1 cache-control header to explicitly mark it as non-cacheable, 4) the document request is a query, 5) the document response has a "pragma: no cache" header, or 6) the document request method is other than GET. We did not consider documents carrying a Set-Cookie header as non-cacheable, since the HTTP 1.1 protocol, as opposed to the HTTP 1.0 protocol, allows these responses to be cached. The percentage of non-cacheable objects that we have found in EURECOM's cache has also been confirmed by other studies [85] [65]. The sum of first-access misses, capacity misses, and update misses (*First-Access+Capacity+Update*) account for as much as 50% of all requests. We expect capacity misses to be a very small fraction of these requests since EURECOM's cache has a 6 GB disk space, which is enough for EURECOM's small client population [79]. The percentage of update misses (*Update*) is about 9%, therefore, the first-access misses account for a big percentage of the misses (30%-50%). The number of first-access misses is very high is because EURECOM's cache has a very small and diverse client population.

Next, we calculate the *maximum achievable hit rate (Max-HIT)* assuming an ideal scenario where institutional caches have an infinite storage capacity and all documents are prefetched in the cache before they are requested. In this ideal case, all requests are hit in the institutional cache, except requests for non-cacheable documents and requests where the client imposes a poll check with the origin server (i.e., reload). This scenario is equivalent to having a huge number of institutional caches cooperating via a satellite distribution where the probability that a client is the first client asking for a document is very small and thus first-access misses, and update misses become hits. The *Max-HIT* at EURECOM's cache is about 85%. Therefore, if EURECOM's cache would be connected to a cache-satellite distribution with many cooperating institutional caches the hit rate could improve by a maximum of 30% to 50%.

4.6.2 Cache-Satellite Distribution Study

To investigate the effect of a cache-satellite distribution we analyzed the logs of an ISP in the USA (AYE [11]), which is connected to a real cache-satellite distribution [83]. We took one week of logs (Dec 18-25, 1998), which account for $13,689,620$ requests. This ISP gives access to about 1000 residential and business clients and has an institutional cache with 48 GBytes of disk space that is connected to the SkyCache [83] satellite distribution. SkyCache's cache-satellite distribution was joining at the time the logs were collected more than 30 ISPs in the USA, including several national ISPs. SkyCache was pushing new documents and document updates at a rate of 4 Mbps twenty four hours a day. The master center of the SkyCache satellite distribution was only broadcasting those documents that received more than 2 or 3 requests in an update period, depending on the available satellite bandwidth [73].

Analyzing the log traces from AYE's cache we could identify hits produced by documents pushed through the satellite channel, and hits produced by documents previously fetched by local clients, which are independent of the cache-satellite distribution. Thus, we could calculate the hit improvement in AYE's cache achieved by the cache-satellite distribution. Figure 4.8 shows that the total *HIT* rate at AYE's cache if the cache would not be connected to the satellite distribution is 30%-40%. The number of first-access misses is much smaller than in the EURECOM's trace, 25%-35%, since AYE's client population is much higher than EURECOM's. When AYE's cache is connected to the satellite distribution there are many requests that are satisfied by documents previously pushed through the satellite distribution. The increase in the hit rate offered by a cache-satellite distribution in AYE's cache is about 18%-25%, thus, AYE's cache is having a total local hit rate (*HIT+Satellite*) of about 60%. The maximum possible hit rate (*Max-HIT*) assuming the idealized case where all documents are prefetched into the institutional caches and caches have infinite disk is about 90%.

4.6.3 Scaled Satellite Distribution

Now, we proceed to simulate the case where the cache-satellite distribution network of which AYE's cache is part, has a much larger client population. For this purpose we took seven days of logs from the four major parent caches, "bo", "pb", "sd", and "uc" in the National Web Cache hierarchy of the National Lab of Applied Network Research (NLANR) [6] from Dec 18th to 25th, 1998 ($32,892,307$ requests). These caches are the top-level caches of the NLANR caching hierarchy and all cooperate to share their load. We simulate the scenario where we have the satellite distribution described in the previous section and then we assume that NLANR top-level caches get also connected to the satellite distribution.

To simulate this new scenario, we consider that AYE's cache is connected to the SkyCache network. Then, for every request in the AYE's trace that results in a miss, we check if the request could be satisfied by any of the documents requested in the NLANR trace. If some client in the NLANR caches previously accessed the document missed in AYE's cache, the document would have been distributed through the satellite and would result in a local hit in AYE's cache. To model a more realistic situation, we consider that the AYE cache is finite and can only keep a certain number of documents pushed through the satellite from NLANR (i.e., one day). This corresponds to a storage capacity of 16 Gbytes, which have to be added to the 48 GB that AYE's cache already had.

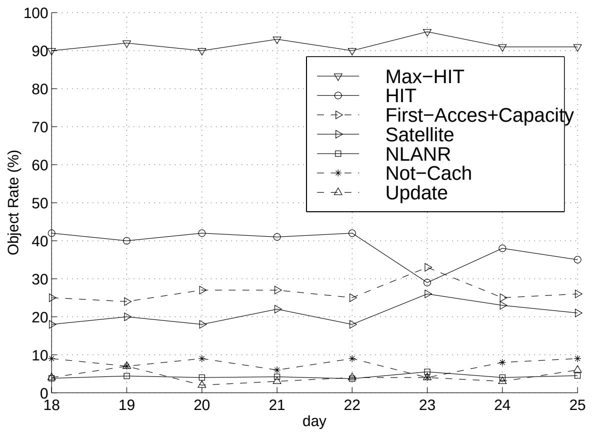


Figure 4.8: General distribution of AYE's cache log entries. Hits are shown with a continuous line, Misses are shown with a dashed line. NLANR entry shows the additional hit rate achieved when NLANR top-level caches also connect to the satellite distribution.

In the absence of MIME headers (i.e., last-modified, expires, etc.) in both, NLANR or AYE traces, it is difficult to ensure that a document miss in AYE's cache, whose URL matches one entry in the NLANR logs, is still up-to-date. To verify if two document requests with the same URL correspond to the same document version, we compared the documents size of the request in the AYE logs and in the NLANR logs. Thus, if the URL and the document size of a miss in the AYE logs matches the URL and the document size of any entry during the previous 24 hours of the NLANR logs, we account for a new document hit.

In Figure 4.8 we show the increase in the hit rate at AYEs institutional cache when NLANR caches get also connected to the satellite distribution. The additional hit rate offered by the NLANR traces is 4% to 5%, resulting in a total hit rate (*HIT+Satellite+NLANR*) of about 65%. The benefits achieved by adding the NLANR logs are not very high since AYE's cache is already connected to a cache-satellite distribution with a large effective client population. When the number of caches connected by the satellite is already large, adding a new cache to the cache-satellite distribution offers only minor improvements. In the same way, when ISPs with a large client population connect to the cache-satellite distribution, the expected hit rate improvement is small. However, the satellite distribution helps reducing the bandwidth costs of any ISP, since documents are prefetched into the caches using the satellite link, freeing the expensive Internet terrestrial connections.

Hit rates close to 65% have also been reported in a caching hierarchy [79][85]. However, in a caching hierarchy a request needs to travel up in the network through a series of caches until the document is hit. These caches can be very congested or can be connected trough very congested links, in which case clients can experience high response times. Instead, a cache-satellite distribution can achieve the same hit rates locally with fast response times, low bandwidth usage, and a minimum configurations. However, there is still a big gap between the 65% hit rate and the maximum achievable hit rate 90%. This can be due to the fact that the local cache is only preloaded with one day of logs from the NLANR caches; if more days would be used the hit rates would increase. Also, we should note that a satellite distribution needs to receive the first request for a document before it is transmitted through the satellite. If the number of documents that are requested only once is high, the maximum achievable hit rate is limited (see Section 4.5.2).

4.7 Conclusions

Caching is being extensively deployed in the Internet to alleviate the problems related to the exponential growth of the Web. However, caching has a limited performance due to the high number of requests not hit in the cache. Requests not hit in a cache are requests for documents that have been purged from the cache, requests for non-cacheable documents, or requests for documents that no client has requested before. Since the price of the disk storage is dropping very fast, we expect that only few documents will be purged from the caches due to space constraints. Only when the number of multimedia documents and continuous media streams grows, limited disk capacity may become a problem again. Non-cacheable documents may become cacheable if some intelligence is placed in the caches to cope with dynamic content [22]. Therefore, a large number of requests not hit in the cache are documents that *no client has requested previously* (first-access misses). To reduce the number of requests in a cache that result in first requests for a document, a cache needs to be shared by a large client population.

In this chapter we analyzed a cache-satellite distribution that interconnects many caches and creates an effective client population that is very large. A cache satellite distribution does not require any inter-cache cooperating protocol and drastically reduces the number of requests that are first requests for a certain document. We have presented analytical models and performed trace-driven simulations to evaluate the performance and requirements of a cache-satellite distribution in terms of hit rate, disk space at the caches, document-retrieval latency, and bandwidth needed at the satellite. We have found that small ISPs with a cache disk space of about 64 GB connected to a cache-satellite distribution, can improve their local hit ratios by 25% to 35%. For ISPs with a large client population (e.g., several thousand clients), using a cache-satellite distribution, offers only marginal improvements in the hit rate. However, ISPs can significantly reduce the cost of filling their caches when using a single satellite channel to fill all ISP's caches, freeing up more expensive Internet terrestrial links for other services. In the case that home clients are also connected to the satellite distribution, they will not suffer the last mile problem for the Web traffic since documents are pushed directly to their disks. As a last remark, one should notice that a cache-satellite distribution appears as a very effective way to easily reduce the number of requests that are first request for a certain document. However, if the number of non-cacheable documents increases, the performance of Web caches will be clearly reduced and the importance of a cache-satellite distribution or any other cache-sharing protocol will be less relevant.

Chapter 5

SPREAD: Scalable Platform for Content Distribution

In Chapter 2 and 3 we showed that a proxy caching architecture can mimic an *application-level multicast* distribution, reducing the bandwidth usage in the network, the load at origin servers, and also client latency. In this chapter, we introduce SPREAD – a new architecture of proxies for distributing and maintaining up-to-date Web content that simultaneously employs three different mechanisms: client validation, server invalidation, and replication. Proxies within SPREAD self-configure themselves to form scalable distribution hierarchies that connect the origin servers of content providers to clients. Each proxy autonomously decides on the best mechanism based on the object's popularity and modification rates. Requests and subscriptions propagate from edge proxies to the origin server through a chain of intermediate proxies. Invalidations and replications travel in the opposite direction. SPREAD's network of proxies automatically reconfigure when proxies go down or come up, or when new ones are added. The ability to spontaneously form hierarchies is based on a modified Transparent Proxying mechanism, called *Translucent* Proxying, that sanitizes Transparent Proxying. It allows proxies to be placed in an ad-hoc fashion anywhere in the network - not just at *focal* points within the network that are guaranteed to see *all* the packets of a TCP connection. In this chapter we (1) describe the architecture of SPREAD, (2) discuss how proxies determine which mechanism to use based on local observations, and (3) use a trace-driven simulation to test SPREAD's behavior in a realistic setting

5.1 Introduction

Due to the explosive growth of the World Wide Web, Internet Service Providers (ISPs) throughout the world are installing proxy caches to reduce user perceived latency as well as bandwidth consumption. Such proxy caches are under the control of the ISP, and usually cache content for its client community, irrespective of the origin server. These proxy caches are often called *Forward* proxy caches to distinguish them from *Reverse* proxy caches, which we discuss next.

More recently, several companies, such as Akamai [2] and Sandpiper [42] have begun offering proxy-based solutions to Content Providers, as opposed to ISPs. The business model is based on the observation that improving a user's browsing experience is not only in the ISP's interest, but in the Content Provider's interest as well. This is becoming increasingly important, as the number of Content Providers keeps growing, and competition for the attention of end users becomes more and more frequent. Proxy caches used in such a scenario are often called *Reverse* proxy caches, to underline the fact that they are controlled by and represent the interests of the Content Provider (or its agent). Reverse proxy caches serve content on behalf of the Content Provider, usually to any arbitrary client on the Internet.

SPREAD can be realized in both the forward proxying and reverse proxying contexts. In this chapter we consider the forward proxying context. Applying SPREAD in a reverse proxying context would need minor alterations, which we indicate at various points in the chapter.

5.1.1 Object Consistency

One of the tenets of SPREAD is that it provides *strong* object consistency. This means that content served to the clients is the same content that is stored at the origin servers at every point in time. Technically it is impossible to guarantee such strong consistency, since there is a non-zero delay between the time a proxy cache receives an object from an origin server, and the instant it serves it to a client. We assume that the update time of a document is much larger than the delay between origin servers and clients, thus, the probability that a document is updated while it is being transmitted from the origin server to the clients is negligible.

The term *strong* is generally used to distinguish from *weak* consistency schemes, which use heuristics to determine the time-to-live of a document but do not provide consistency guarantees. We believe strong consistency is imperative, especially now that people have begun to rely on the Web in timely information for conducting business, and because an increasing number of sites have begun to offer time-sensitive information.

Forward proxies have been known to be notoriously sloppy in this area. While mechanisms exist within the HTTP protocol for maintaining cache consistency, in practice, forward proxy caches administered by ISPs, use their own time-to-live (TTL) heuristics [47], [88] that are engineered in a rather arbitrary fashion. Historically, part of the effect (or some say the cause) has been that Content Providers often misuse or abuse features of the HTTP protocol, using techniques such as *cache-busting*. Regardless of how one sees this tension between ISPs and Content Providers, we believe that adhering to strong consistency mechanisms in accordance with the HTTP guidelines is important. This is a fundamental design guideline in SPREAD.

SPREAD uses three primary mechanisms to achieve strong consistency:

- Client Validation *(V)*: In this mechanism, for every client request that a proxy receives the proxy always checks back with the origin server to see if the object copy is fresh. This is typically accomplished by an *If-Modified-Since (IMS)* HTTP Request. If the origin server finds the object in the proxy fresh, the proxy cache will respond to the client with its cached copy. If the object has expired, the client will receive the master object from the origin server and the cache will keep an object copy. The only exception to this rule is if the object has been explicitly marked as cacheable, and a Max-age, Expires, or an equivalent piece of metadata has been set to a value by the origin server that indicates a non-zero time-to-live (TTL). In such a situation the proxy will not need check back with the server to validate the cached copy of the object for the stipulated TTL.

- Server Invalidation *(I)*: With invalidation, a proxy cache first subscribes to an invalidation service for that object (or range of objects) with the origin server, or an agent for the origin server that is responsible for signaling the expire of the object. In this case, the proxy cache assumes that the object is fresh unless an invalidation message from the origin server is received by the proxy to explicitly expire the object. Using invalidation, the first client request after the object is invalidated experiences high latency since the object needs to be retrieved from the origin server.

- Replication *(R)*: With replication, updated versions of the object are explicitly preloaded in the proxy cache by using *push*, or equivalently a *pseudo-push* that can be implemented with a periodic-pull. As with invalidation, a proxy cache must express interest in the object (or a range of objects) a-priori, by subscribing to the replication service. Using replication, clients always experience very small latencies, however, the bandwidth consumed can be wasteful in cases where there are more updates than requests.

To save on bandwidth, instead of sending the entire object, one may send just the *diff*, or some encoded form of the *change* between the old and new versions. This may be applied to all of the above mechanisms. The results of this chapter remain valid under such a scenario as well.

5.1.2 Our Approach

A novel feature of SPREAD is that its proxies *dynamically* choose between client invalidation, server invalidation, and replication, on a per-object basis. This is discussed in detail in Section 5.4. Earlier studies [90], [48] have analyzed the benefits of server invalidation versus client validation, but their comparisons were in a context where strong cache consistency was not imperative. More importantly, their evaluation had been focused on assessing stale hit-rate using trace-driven simulations at a *macroscopic* level. In our work, we evaluate the competing mechanisms to keep strong consistency from a more fundamental perspective, analyzing the problem at the level of individual reads and writes of each object, which we believe yields substantial insight. The authors of [58] propose unicast invalidations instead of adaptive time-to-live mechanisms to keep strong consistency, however, using unicast communication from the server to the clients makes their approach non-scalable. The authors of [93] study the efficacy of server invalidations using a scalable distribution infrastructure, and provide several insights into the general problem of cache consistency. Our work advances the state-of-the-art beyond [93] in three major respects. First, proxies in our system *dynamically* choose the consistency mechanism based on their own observation of the request rates and update rates of objects. Prior knowledge of these statistics is not assumed. Second, our analytic results help us define the thresholds of the optimal control policy at which proxy caches switch from one mechanism to another, which in turn helps us in building a smarter overall system. A third novel feature of SPREAD is its ability to spontaneously build content distribution hierarchies, without prior knowledge of the existence of other proxies. If a proxy lies along the natural path from an edge proxy to the origin server, it intercepts communication between them. Communication includes Web requests, as well as subscriptions for invalidation and replication. Such incremental actions by intermediate proxies builds sophisticated multi-level hierarchies rooted at origin servers. The interception is at the TCP layer. While the possibility of using Layer-4 transparent proxying for building hierarchies has been considered in [54], the scope of such an architecture is limited because of the problem that all packets of a TCP connection may not always follow the same path. If a transparent proxy intercepting a connection is unable to see all the packets of the connection, it cannot sanely proxy the TCP connection, which is a well known limitation [28]. A partial solution is to deploy transparent proxies at *focal points* within the network, which are guaranteed to see all packets of a connection. This makes the ad-hoc placement of proxies infeasible. SPREAD solves this problem by using what we call *Translucent* proxying, which guarantees that a proxy that sees the SYN of a TCP connection, will see all subsequent packets as well. This is accomplished by a novel use of IP tunneling and TCP-OPTIONS which we will discuss later.

5.2 SPREAD Architecture

The SPREAD architecture is based on a scalable content distribution network that spontaneously builds proxy caching hierarchies. In SPREAD, edge proxies connect to servers using a chain of proxies that are on the natural path from the edge proxy to the origin server (see Figure 5.1). Any given edge proxy may be a part of multiple proxy caching hierarchies rooted at different origin servers. In this section we discuss the basic principles that enable this. Unless otherwise mentioned, we assume a forward proxying scenario.

It is important to note that SPREAD is not concerned with how clients reach edge proxies. This is considered orthogonal to SPREAD. While this is indeed a non-issue in the case of forward proxies (which typically have a fixed or long-term mapping of clients to edge proxies), the reverse proxy scenario is trickier. With the advent of dynamic DNS tricks, the mapping of clients to edge proxies is becoming relatively easy.

5.2.1 Self-Configuring Caching Hierarchies

A proxy caching hierarchy acts as an *application-level multicast* distribution tree (Chapters 2 and 3), reducing the bandwidth usage in the network, the load at origin servers, and also reducing client latency. In the absence of a proxy caching hierarchy, origin servers need to directly communicate with all edge proxies, creating a huge burden on the origin server and the network. Using IP-multicast between the origin server and edge proxies would require an infeasible large number of multicast groups, and in addition IP-multicast is not widely deployed.

Proxy caching hierarchies already exist in the current Internet [6]. However, current hierarchies are static and require substantial manual configuration and maintenance. To generate caching hierarchies that automatically configure themselves and forward packets to the origin servers through the shortest path routes, a routing architecture at the application level can also be implemented. Caches would then exchange application-level costs and calculate

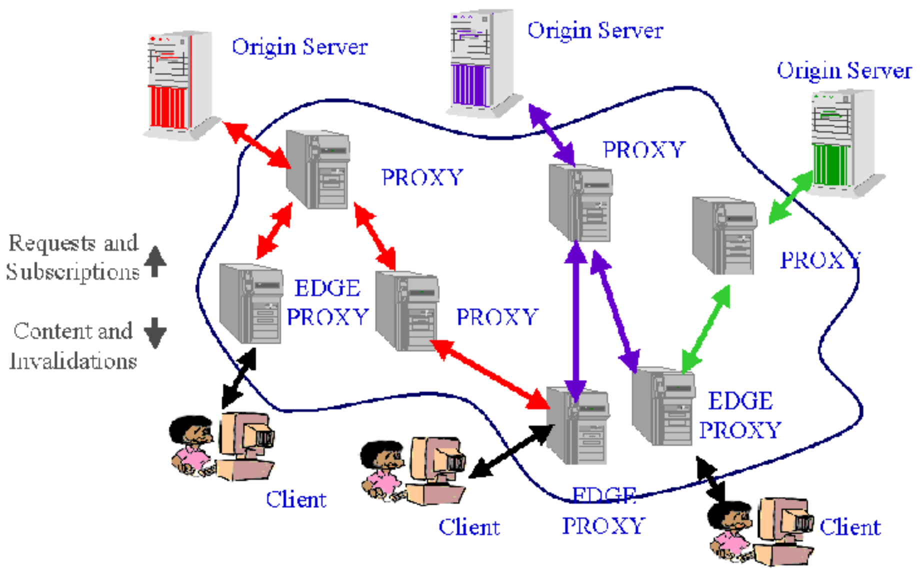


Figure 5.1: SPREAD Architecture

the best path to every origin server [67]. However, building an application level routing infrastructure is non-trivial, since route changes in the underlying network layer, will impact application-level routing. In contrast, SPREAD uses network layer routing and transparent proxies to build its proxy caching hierarchies. Requests travel from the clients to the origin servers following the shortest network path, and intermediate transparent proxies automatically pick up the connections for Web traffic (port 80). A transparent proxy that picks up a connection directly satisfies the document request if the document is stored in its cache, or lets the request travel towards the origin server if the document is not stored in its cache. As the request travels towards the origin server, the document request may be intercepted again by other transparent proxies, automatically forming a caching hierarchy. Changes in routes will create new hierarchies spontaneously, which will obey network level routing. No extra signaling is required to maintain the hierarchy.

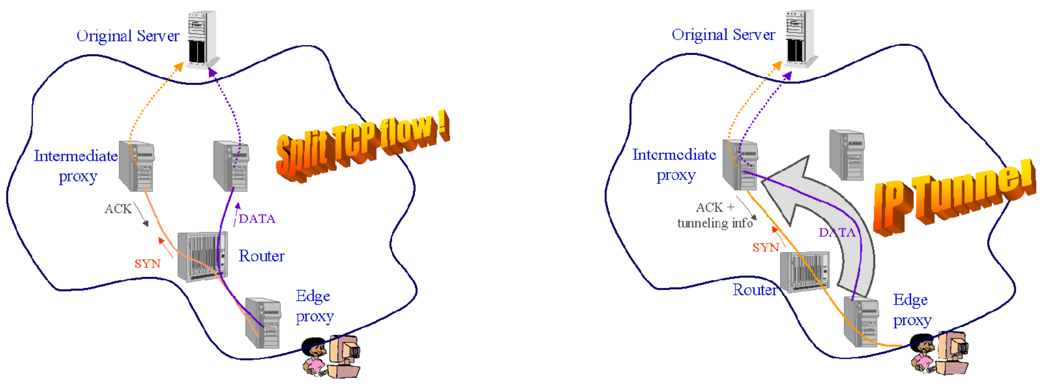
Naively building a hierarchy using transparent proxies is elegant, but has a serious problem. Since routing in an IP network can lead to situations where multiple paths from client to server may have the same lowest cost, it can happen that packets of a connection follow multiple paths. In such a situation, a transparent proxy may see only a fraction of packets of the connection. Occasionally it is also possible that routes change mid-way through a TCP connection, due to routing updates in the underlying IP network. This problem limits the scope, requiring transparent proxies to be deployed exclusively at the edges or *focal* points within the network where they are guaranteed to see all the packets of the connection. SPREAD addresses this limitation by using Translucent Proxying, which allows the placement of proxies *anywhere* in the network.

5.2.2 Translucent Proxying

Translucent Proxying Of TCP (TPOT) is a more sophisticated transparent proxying mechanism that allows proxies to be cascaded and networked together transparently, eliminating split TCP flows. Figure 5.2(a) provides a high level overview of the problem of split TCP flows and how Translucent Proxying solves the problem. When an edge proxy intends to connect with an origin server as shown in Figure 5.2(b), it issues a SYN packet, which reaches the intermediate proxy on the left. If the next packet of the TCP connection should be routed towards the proxy on the right, we have a situation where the proxy on the left cannot properly proxy the TCP connection. In Translucent Proxying, the proxy on the left sends back in the ACK, a signal to the edge proxy providing its IP address. The edge proxy will then use the IP address, to *tunnel* all remaining packets via the proxy on the left.

Before describing the TPOT protocol, we provide a brief background of TCP/IP, which will help in better understanding TPOT.

Each IP packet typically contains an IP header, and a TCP segment. The IP header contains the packet's source and destination IP address. The TCP segment itself contains a TCP header. The TCP header contains the source port and the destination port that the packet is intended for. This 4-tuple of the IP addresses and port numbers of the source and destination uniquely identify the TCP connection that the packet belongs to. In addition, the



(a) Without Translucent Proxying. (b) With Translucent Proxying.

Figure 5.2: Translucent Proxying solves the Split Flow problem using IP Tunneling.

TCP header contains a flag that indicates whether it is a SYN packet, and also an ACK flag and sequence number that acknowledges the receipt of data from its peer. Finally, a TCP header might also contain TCP-OPTIONs that can be used for custom signaling.

In addition to the above basic format of an IP packet, an IP packet can also be encapsulated in another IP packet. At the source, this involves prefixing an IP header with the IP address of an intermediate tunnel point on an IP packet. On reaching the intermediate tunnel point, the IP header of the intermediary is stripped off. The (remaining) IP packet is then processed as usual.

We now describe the TPOT protocol. Consider a source S that intends to connect with destination D via TCP, as shown in Figure 5.3. Assume that the first (SYN) packet sent out by S to D reaches the intermediary TPOT proxy T. (S,S_p,D,D_p) is the notation that we use to describe a packet that is headed from S to D, and has S_p and D_p as the source and destination ports respectively.

To co-exist peacefully with other end-points that do not wish to talk TPOT, we use a special TCP-OPTION "TPOT," that a source uses to explicitly indicate to TPOT proxies within the network, such as T, that they are interested in using the TPOT mechanism. If T does not see this option, it will take no action, and simply forwards the packet on to D on

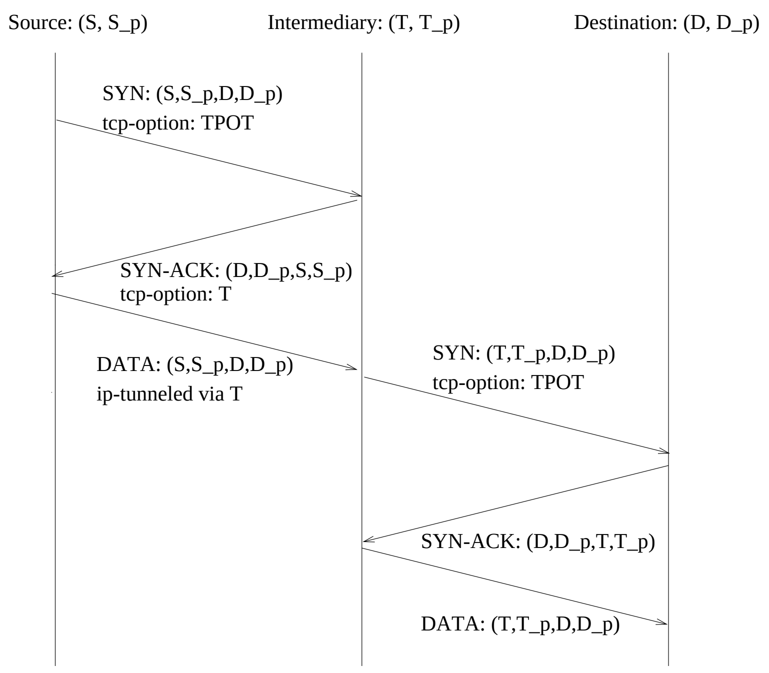


Figure 5.3: The Translucent Proxying Protocol

its fast-path. If T sees a SYN packet that has the TCP-OPTION “TPOT” set, it responds to S with a SYN-ACK that encodes its own IP address T in the TCP-OPTION field. On receiving this packet, S must then send the remaining packets of that TCP connection, IP tunneled to T. From an implementation standpoint this would imply adding another 20 byte IP header with T’s IP address as destination address to all packets that S sends out for that TCP connection. Since this additional header is removed on the next TPOT proxy, the total overhead is limited to 20 bytes regardless of the number of TPOT proxies intercepting the connection from the source to the final destination. This overhead can be further reduced by IP header compression [30] [50].

In SPREAD we use TPOT both for regular HTTP Requests as well as for subscriptions and unsubscriptions. Consider the case of a regular HTTP Request. For a cache hit, T is able to satisfy a request from S, and the response is simply served from one or more caches attached to T. In the case of a cache miss, T communicates with the destination D as shown in Figure 5.3. Note that the proxy T sets the TCP-OPTION “TPOT” in its SYN to D to allow possibly another TPOT proxy along the way to again proxy the connection. In Figure 5.3 we do not show such a scenario.

A more comprehensive description of the TPOT protocol, its variants, scalability and performance issues, as well as a prototype implementation may be found in [77].

5.3 Automated Content Distribution

In this section we describe basic content distribution in SPREAD. Edge proxies request objects from origin servers and requests are transparently intercepted by intermediate translucent proxy caches en-route to the origin server.

Proxies periodically calculate the expected number of requests per update period for every object, or for a volume (set of objects). Depending on the number of requests per update period, proxies may subscribe to invalidation or replication (see Section 5.4). As the subscription travels to the origin server, an intermediate translucent proxy en-route intercepts the subscription (unless the intermediate proxy is overloaded - in which case it lets the subscription pass through). On intercepting a subscription for invalidation, the intermediate proxy will subscribe itself to such a service, which in turn may be re-proxied by yet another proxy. Note that it is possible to limit this recursion by adding a hop-count field to the subscription, which gets decremented at each proxy. Once the counter hits zero, no other proxy will intercept it.

As we will see later, one may order mechanisms, in the increasing order V, I, R. If a child proxy finds a certain mechanism optimal, then a parent must, *at least*, use that mechanism. This assumes that children proxies are self-regulating as per SPREAD's optimal control policy. This will be discussed in a later section. In the case where an invalidation (I) subscription arrives at a proxy, the proxy is forced to subscribe itself, unless of course it is already subscribed to I, or to Replication (R) - since R *implies* I. In the case where an R arrives at a proxy, it must subscribe to R, if it is not already subscribed to R. Thus when a child proxy subscribes to I or R, all proxies on the path to the origin server are also automatically subscribed to *at least* that mechanism. Invalidations and replications themselves travel in the opposite direction. When an object is updated at the origin server, the server sends invalidations and/or replicas to proxy caches that are subscribed to I or R. Proxy caches that receive invalidations or replicas will themselves propagate the invalidations or replicas to children subscribed to I or R at the next tier. The process is repeated until the invalidation or document replica arrives at the edge proxy, thus, strong consistency is maintained (see Section 3.4 for a more detailed description of how to send information through a caching infrastructure).

5.3.1 Leases

Subscriptions have leases associated with them. On expiration, a subscription must be renewed. These leases are set large enough so that repeated subscriptions do not overburden the network. At the same time, they are not so large that proxies commit themselves so far into the future when the changing statistics of the request and update rates suggest another mechanism. This is an implementation issue that we do not discuss further.

5.3.2 State Information

Parent proxies need to keep state information about the children proxies that are subscribed to invalidation or replication. However, the amount of state information required to keep track of subscribed children proxies is negligible compared to the disk capacity needed to store objects. Objects are usually subscribed and unsubscribed infrequently, and therefore, the amount of processing required is very small [93]. In addition, if multicasting is used, the load and state information at parent proxy caches is very small since only one object copy needs to be distributed to a set of children proxies (see Section 3.4 for a more detailed analysis of the state information in the caches). To further reduce the load and the state information, objects can be grouped into volumes at the cost of a coarser granularity for optimization and control. Here, a whole volume is invalidated or replicated instead of an individual object.

5.3.3 Reliability and Load Balancing

To ensure strong consistency even in the case of proxy cache failure, parent proxies periodically send *heart beats* to their children proxies. When a parent proxy dies, children proxies set the corresponding objects that the parent was responsible for as stale and re-send subscriptions towards to the origin server. The next (alive) proxies in the path to the origin servers then pick up the new subscriptions and become the new parents. This mechanism makes SPREAD reliable even under outages. A failed proxy or link does not alter the guarantee for strong consistency.

Alternately, when a new proxy appears, it joins SPREAD incrementally. While existing subscriptions are not disturbed (since they are tunneled using TPOT to the existing parent), new subscriptions and Web requests that it sees can be proxied. Existing subscriptions also ultimately get re-proxied once their lease expires.

SPREAD automatically redistributes the load among its proxy caches, since every proxy cache is only responsible for those objects for which it sees requests, and then again only to its children. A last resort for an overloaded proxy server, is simply to stop intercepting any new Web requests and subscriptions, effectively going into *invisible* mode for all future services.

5.4 Automated Mechanism for Strong Consistency

To develop an appreciation for why and how SPREAD may optimize its performance, consider the scenario shown in Figure 5.4. An object is considered hotter than another if it is requested (read) more times than its is updated or modified (written).

Imagine that we want to minimize bandwidth consumption. For objects that are so cold, that every request appears after one or more writes/updates of the object, invalidations are useless, since every new object request sees a new object update. Replication, on the other hand, wastes even more bandwidth since objects are replicated on every write though they are rarely requested. In such a situation, it appears that client validation is probably the best policy. Note that what is important is the relative frequency of reads to writes. Objects that

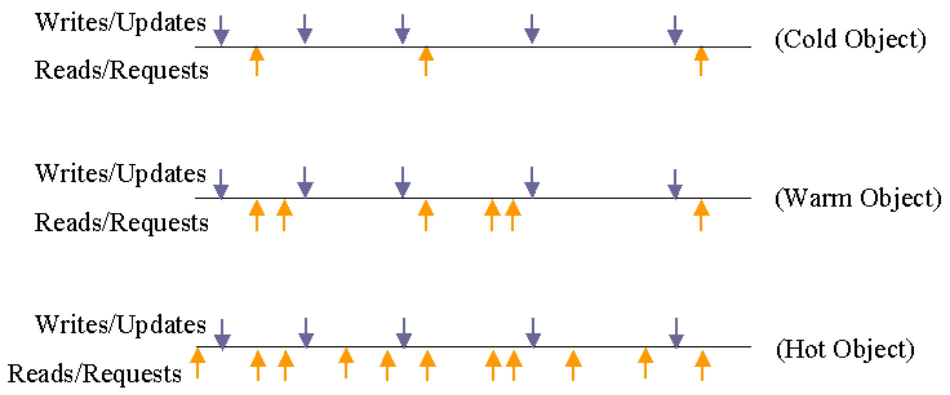


Figure 5.4: Cold, Warm and Hot objects.

are hot, are objects for which there are one or more reads per write. In such a situation, replication is always preferred to client validation. Validation suffers from the problem that the second and future reads in an update/write interval will each require an *If-Modified-Since* poll, even if the object has not changed. The poll consumes bandwidth and causes additional delays. While invalidation performs better than validation, invalidation also wastes some bandwidth due to invalidation messages that perform no constructive function when compared to replication. Indeed, as we will see more rigorously later, invalidation is optimal for warm objects whose frequency of reads/requests is on the same order as the number of writes/updates. Note that in situations where not all three mechanisms are supported by the origin server, SPREAD will simply choose the best from what is available.

5.4.1 Analytical Model

We now build a mathematical framework to investigate how one might formulate the problem of deciding which mechanism to use for a given object. Since these choices will be made at each proxy, the issue of how a proxy estimates the various parameters relating to an object is an important one. These estimation issues will be dealt with in later sections.

We start with the case of an edge proxy that sees requests for some arbitrary object. We will extend our analysis to the case of an intermediate proxy (not just an edge proxy) in later sections. We assume that requests for the object from all the clients connected to an edge proxy cache are Poisson distributed with average request rate λ. We also assume that objects are updated periodically every Δ time units, or following an exponential distribution with average update period Δ, $f(t) = \frac{1}{\Delta}e^{-t/\Delta}$. This assumption will be discussed in a later section (Section 5.4.2).

We denote R_i to be the number of requests for an object i per update period from all clients connected to the edge proxy. In the case when the object is updated periodically, the probability that there is at least one object request per update period from an edge proxy is then given by:

$$P(R_i > 0) = 1 - e^{-\lambda \cdot \Delta}$$

Note that $\lambda\Delta$ is the average number of requests per update period Δ.

When the object is updated following an exponential distribution, the probability that there is at least one object request per update period from an edge proxy is given by:

$$P(R_i > 0) = \int_0^\infty (1 - e^{-\lambda \cdot t}) f(t) dt = \frac{\lambda \cdot \Delta}{\lambda \cdot \Delta + 1}$$

To determine whether to use validation, invalidation, or replication, caches need to estimate the average number of requests per update period on a per-object basis. To calculate the average number of requests per update period, caches need to estimate i) the average request rate of an object and ii) the average update period of an object.

To estimate the request rate of an object, edge proxies can use the access logs from client access. The problem of estimating the request rate for an intermediate proxy is more involved (since it may not see direct hits from clients), and is discussed in Section 5.4.5. It is of course possible for edge proxies to inform intermediate proxies about their request rates (and in fact this was our initial design), but as we shall see later one can do without such communication.

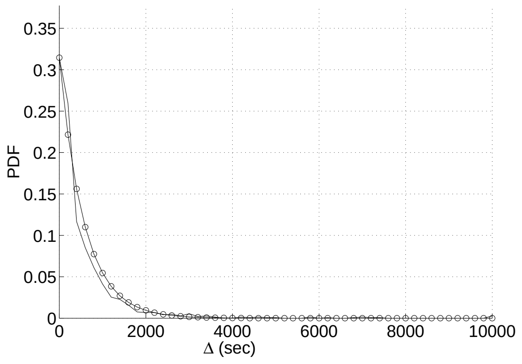
5.4.2 Estimating Update Rate

Proxy caches that are subscribed to invalidation (I) or replication (R) for an object, see all updates, and can therefore estimate the update rate in a straight-forward fashion. However estimating the update period of an object that uses validation (V) is more complex. Since the proxy can only inspect the *Last-Modified* time of an object when it is requested, information on updates that are never requested are lost. However, proxies can use the difference between the time of a request (or Date field) and the Last-Modified time, to infer the average update period of an object if they know the probability distribution of object updates.

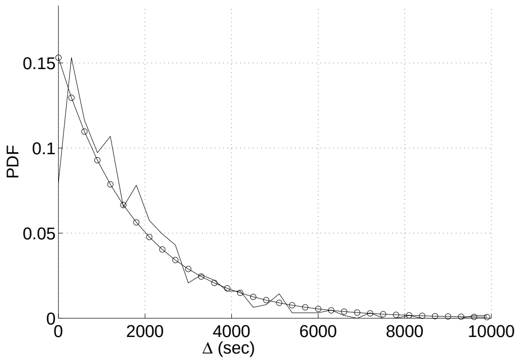
We should point out that headers such as the ”Expires” header which explicitly provide consistency information, cannot be used here for two reasons. First, our own study of the Web and those of others have shown that most cacheable documents have their Expires headers at a value that effectively makes the TTL zero anyway. Further, such protocol headers (even when non-zero) do not provide realistic values for update rates, since, these headers only need to provide a lower bound. In other words, a document whose TTL is set to 10 seconds (via an Expires header or some other metadata) may update itself after 10 days, and yet be perfectly in line with the HTTP protocol.

Previous work on the distribution of object updates suggested that objects are approximately updated randomly following an exponential distribution or periodically [34]. However, these results were performed with client traces that did not see all server updates. To better study the distribution of object updates we polled different sites once every minute for a period of 10 days, recording the last-modified-time stamp of the object on every poll. Then we calculated the update period of an object as the time difference between two different last-modified-time stamps. This experiment gave us the real update pattern of an object within a resolution of one minute. Our results confirm the ones presented in [34]. We found that there are a large number of Web sites that update their documents periodically, e.g. every 15 or 30 minutes. However, we also found a large number of Web sites that update their documents randomly following an exponential distribution. In Figure 5.5 we present the distribution of object updates for two different news sites. We clearly see that the distribution of object updates in both sites approximates an exponential distribution.

Note that proxy caches can easily determine if an object is updated periodically or is exponentially distributed by studying the variance of object updates. Once they have determined if the object is updated periodically or exponentially, they can use the time difference



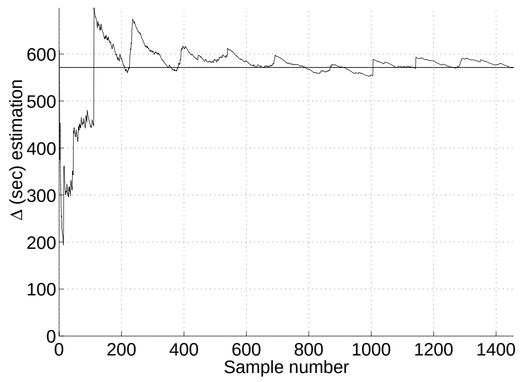
(a) Spanish Newspaper Web Site



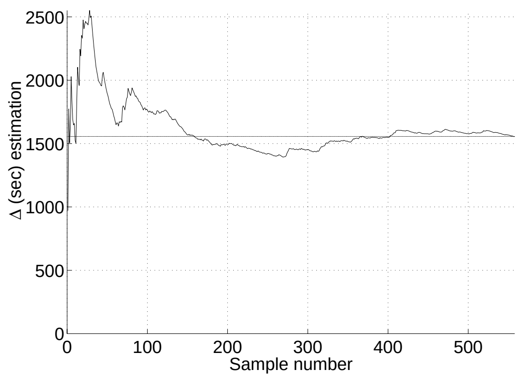
(b) BBC News Web Site

Figure 5.5: Distribution of object update intervals. 10 day logs. Servers are polled every minute.

between object requests and the last-modified-time stamps to estimate the average update period [53].



(a) Spanish Newspaper Web Site



(b) BBC News Web Site

Figure 5.6: Estimation of the average update interval Δ as a function of the number of samples

Figure 5.6 shows how rapidly the estimate of the average update period converges with the number of samples. Each sample measures the time difference between every request and the last-modified-time, and computes a simple average. We observe that after 200 samples, the estimate of the average update period converges to 600 seconds in Figure 5.6(a), and 1500 seconds in Figure 5.6(b). This warm-up time is small enough to make such estimators viable. Note that if proxies do not have enough number of samples to estimate the number of requests per update period of a document, proxies will use *validation* as the default policy.

5.4.3 Optimizing Bandwidth

Next, we compute the bandwidth usage by each mechanism to deliver up-to-date content. We define the bandwidth usage BW as the average number of bytes consumed per update period Δ in a proxy's link.

Let S_o be the actual size of a Web object. Let S_h be the size of an HTTP header, which is considered to be the same as the size of an IMS request. Let S_i the size of an invalidation message.

The bandwidth usage per Web object for validation BW_V, invalidation BW_I, and replication BW_R can be easily shown to be:

$$\begin{aligned} BW_V &= P(R_i > 0) \cdot S_o + \lambda\Delta \cdot S_h \\ BW_I &= P(R_i > 0) \cdot (S_o + S_h) + S_i \\ BW_R &= S_o \end{aligned}$$

Note that in our analysis we have assumed that the object has no max-age or expires header set by the server. In a situation where the server would set a max-age or an expires header different than zero, the analysis would need to be modified accordingly, though the qualitative results of this chapter would still hold.

Figure 5.7 shows the bandwidth usage of validation, invalidation, and replication depending on the average number of requests per update period $\lambda \cdot \Delta$. The values for S_i, S_h and S_o are representative of what is typical for the Web today. For objects with few requests per update period, replication wastes a lot of bandwidth compared to validation or invalidation, since the object is preloaded into the caches even when it is not requested by the clients. On the other hand, validation and invalidation have a low bandwidth usage since the object is

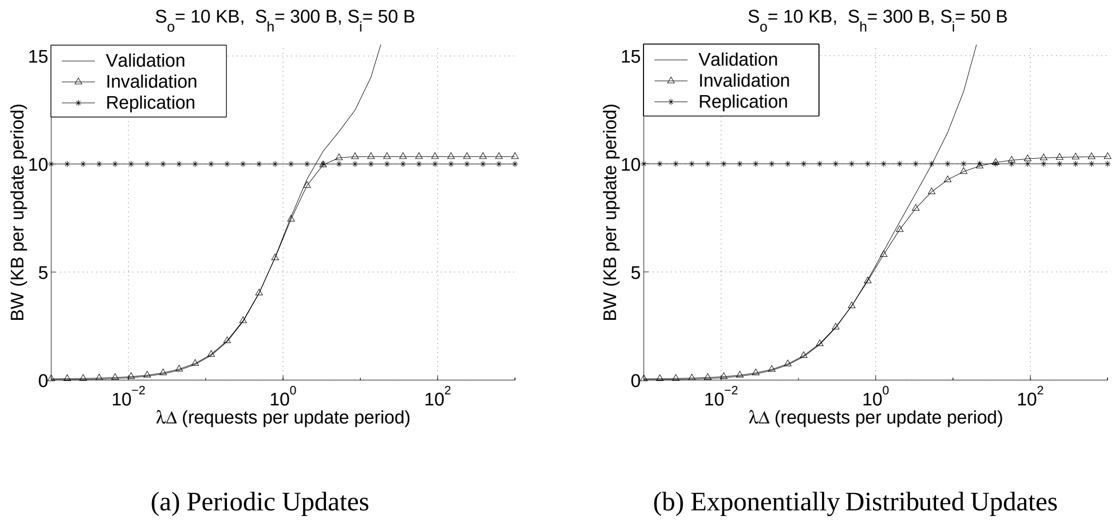


Figure 5.7: Bandwidth usage

only fetched into the caches when it is requested by a client. For a large range of values for $\lambda \cdot \Delta$ from about 0 to 1 requests per update period, validation uses slightly less bandwidth than invalidation since every request finds a new object update and therefore the overhead of IMS requests to the origin server is almost zero. For values above about 1 requests per update period, replication does well, trailed by invalidation, which suffers because of the extra invalidations that are sent out. Validation works poorly, due to the fact that every request generates an IMS request which is typically much heavier than an invalidation. Figure 5.7 shows that the mechanism that consumes the least bandwidth is different in different regimes of $\lambda \cdot \Delta$, and that the order in which the different policies are optimal is V, I, and R as $\lambda \cdot \Delta$ increases.

5.4.4 Switching Thresholds

Let the switching thresholds between V and I, and I and R, be denoted by Th_{VI} and Th_{IR} respectively. Table 5.1 shows the thresholds to switch among the different policies at an edge proxy. Th_{VI} is given by the value of $\lambda\Delta$ that equals the bandwidth usage of validation and invalidation, $BW_V = BW_I$. Similarly Th_{IR} is given by the value of $\lambda\Delta$ that equals the bandwidth usage of invalidation and replication, $BW_I = BW_R$.

Perspective	Th_{VI} (req per update period)	Th_{IR} (req per update period)
Edge Proxy (Periodic)	$\sqrt{\frac{2 \cdot S_i}{S_h}}$	$ln(\frac{S_o+S_h}{S_h})$
Edge Proxy (Exponential)	$\frac{S_i+\sqrt{S_i^2+4S_hS_i}}{2 \cdot S_h}$	$\frac{S_o-S_i}{S_h+S_i}$

Table 5.1: Thresholds to switch between validation and invalidation Th_{VI}, and between invalidation and replication Th_{IR}

From Table 5.1 it is easy to prove that for all reasonable values of S_i, S_h and S_o, we have the property that: $Th_{VI} < Th_{IR}$. Further, since λ increases as one moves closer to the origin server, we have the property that at any level of the hierarchy if a given mechanism is optimal for a proxy, it must be *at least* good for the parents above. That is:

- if a proxy finds V optimal, then its parent may find V, I, or R optimal.
- if a proxy finds I optimal, then its parent may find I, or R optimal.
- if a proxy finds R optimal, then its parent will find only R optimal.

By the above result, if a proxy subscribes to a certain policy, it must also be in its parent's best interest to *at least* have that policy in place. Therefore it always makes sense to proxy subscriptions on behalf of a child proxy. This clearly validates SPREAD's design model, even if by serendipity.

5.4.5 Estimating Request Rate at an Intermediate Proxy

As we discussed earlier, estimating the request rate of an object at an intermediate proxy may be complicated because it does not see direct hits from clients. However, we argue here that given the observations of the previous section, this can be substantially simplified by breaking down the possibilities into two cases.

- Case 1: If *any* of the children are in the R state, then, the parent proxy is also in the R state and cannot go to I until all of its children unsubscribe from R. No decision need be made by the proxy, and therefore estimating request rate is not essential. (Note that when the last child proxy unsubscribes from R, we can seed the estimator with the estimated request rate from that child to be $\frac{Th_{IR}}{\Delta}$).

- Case 2: In this case, children proxies are in the I or V state. For those in the V state the estimation of the request rate is straightforward, since the proxy sees all the requests (HTTP GETs or IMS requests). For proxies in the I state, the request rate may be computed in a more sophisticated fashion. Here, the proxy estimates the time interval between an invalidation and the immediate following request. For both exponentially distributed and periodic (deterministic) update periods, we may compute an estimate for the request rate from that child proxy using standards results for residual life from the area of Renewal Theory [26].

5.4.6 Latency

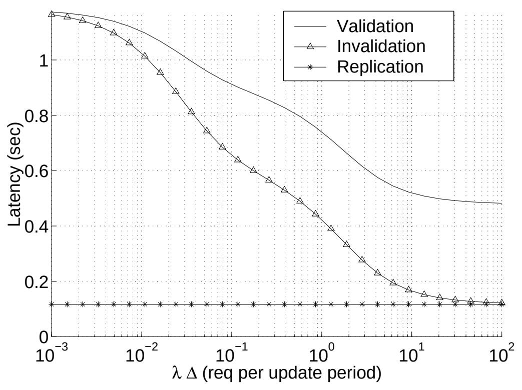
In this section we investigate the latency experienced by the clients when validation, invalidation, or replication are used. Let T_{os} be the transmission time of an object when it is retrieved from the origin server. Let T_{pc} and T_{cc} be the transmission time of an object when it is transmitted from the parent proxy and from the children proxies respectively. Let RTT_{os} be the round-trip-time between the origin server and any proxy cache. The expected latency experienced by a client depends on the tree level where the object is hit. Let L be the number of links traversed to find a object. In this section, we consider a simple two-tier caching hierarchy with a certain number of children caches (cc), one parent cache (cc) and one origin server (os), however, the analysis can be easily extended for a different number of cache tiers. The exact calculation of the probability distribution function of L is the same than the one in Section 2.4.1. Given the distribution of L we can calculate the expected latency experienced by a client for validation T_V, invalidation T_I, and replication T_R as:

$$T_V = P(L = cc) \cdot (T_{cc} + RTT_{os}) + P(L = pc) \cdot (T_{pc} + RTT_{os}) + P(L = os) \cdot T_{os}$$

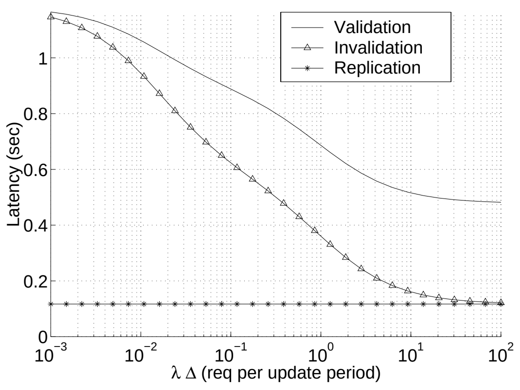
$$T_I = P(L = cc) \cdot T_{cc} + P(L = pc) \cdot T_{pc} + P(L = os) \cdot T_{os}$$
$$T_R = t_{cp}$$

To consider real values for the latency, we analyzed 10 days of logs on the local proxy at Eurecom, which is connected to a caching hierarchy through a parent proxy. We averaged the latencies during the 10 days of the trace to obtain the following values:

- Transmission time from the local proxy: T_{cc} = 117 msec,
- Transmission time from a parent proxy: T_{pc} = 585 msec,
- Transmission time from the origin server: T_{os} = 1183 msec,
- Round-Trip-Time to the origin server: RTT_{os} = 300 msec.



(a) Periodic Updates



(b) Exponentially Distributed Updates

Figure 5.8: Expected Latency

We considered the case of a caching hierarchy with 64 children caches and a single parent cache. Based on these values Figure 5.8 shows the latency experienced by a client for validation, invalidation, and replication. Using replication, clients always experience small latencies since the edge proxy always has the object replicated to it. This, as we have seen earlier, may be extremely wasteful of bandwidth. As the number of requests per update period increases, the probability of finding an object at proxies closer to the client increases, thus, reducing the latency experienced. Invalidation offers better latencies than validation since client requests do not need to contact the origin server every time. However, for invalidation to provide similar latencies as replication, the number of requests per update period needs to be very high (i.e. approx. 100 requests/update period). For such popular objects, using invalidation to reduce client's latency is not the best option since replication generates slightly less traffic in the network (see Figure 5.7), providing very small latencies for *all* receivers.

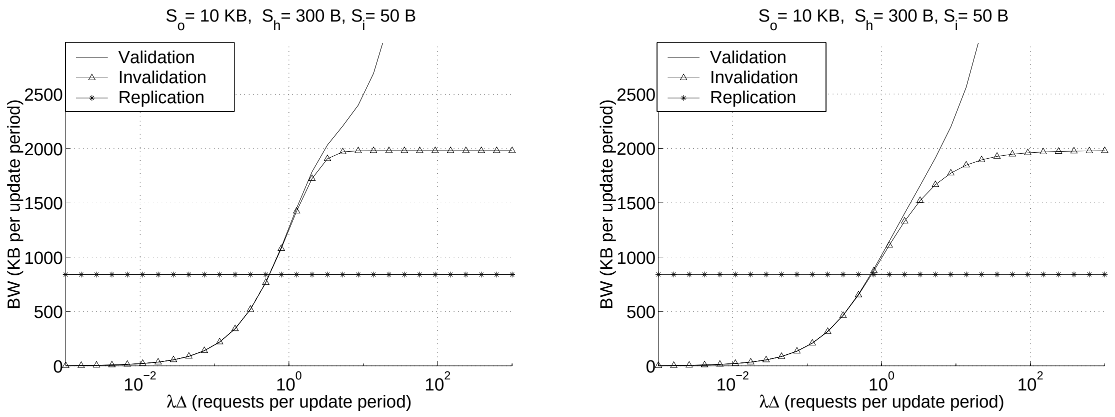
5.4.7 Multicast Extensions

In this section we consider the case when the network among proxies supports multicasting. If multicasting is available, parent proxies may decide to multicast invalidations and replicas to their children proxies instead of sending them via unicast. For validation, objects and IMS messages are distributed via unicast. For invalidation, the actual object is fetched via unicast by the children proxies, however, invalidation messages are multicast to all proxy caches. For replication, object updates are pushed via multicast from the parent proxy cache to all children proxies.

The decision to use multicast or unicast depends on the multicast gain $G = \frac{C_{mc}}{C_{uc}}$, that is the multicast cost C_{mc} divided by the unicast cost C_{uc}, which is a function of the network topology, the number of children proxies and their location. Several studies have shown that the multicast gain in a wide range of network topologies can be approximated by G=$M^{-0.2}$, where M is the number of receiving proxies [71]. Therefore, it is enough for a parent proxy to know the number of subscribed children proxies to estimate the multicast gain and therefore decide whether to turn on multicast or not (see Section 3.5 for a more detailed analysis of when to turn multicast on).

The bandwidth usage in the network of validation BW_V, invalidation BW_I, and replication BW_R to deliver one byte from a parent proxy to the children proxies with multicast is:

$$\begin{aligned} BW_V &= P(R_i > 0) \cdot S_o \cdot C_{uc} + \lambda\Delta \cdot S_h \cdot C_{uc} \\ BW_I &= P(R_i > 0) \cdot (S_o + S_h) \cdot C_{uc} + S_i \cdot C_{mc} \\ BW_R &= S_o \cdot C_{mc} \end{aligned}$$



(a) Periodic Updates (b) Exponentially Distributed Updates

Figure 5.9: Bandwidth usage with multicast enabled

To study the effect of a multicast distribution we consider that the network connecting the parent proxy with its M children proxy caches is a full O-ary tree with height H. In Figure 5.9 we present the bandwidth usage inside the network for validation, invalidation, and replication when multicast is enabled.

Comparing Figures 5.7 and 5.9 we observe that the relative performance of validation is not modified since validation does not benefit from the fact that multicast is enabled. We also observe that the relative performance of invalidation is slightly smaller since invalidation messages are now multicasted. For replication, the bandwidth savings are very high, since the cost of replication is small. Of course, the multicast gain depends on the network topology and the number of receivers; however, even in the worst case a multicast distribution performs no worse than unicast, and the relative performance of validation, invalidation, and replication would then be the same as the one in Figure 5.7.

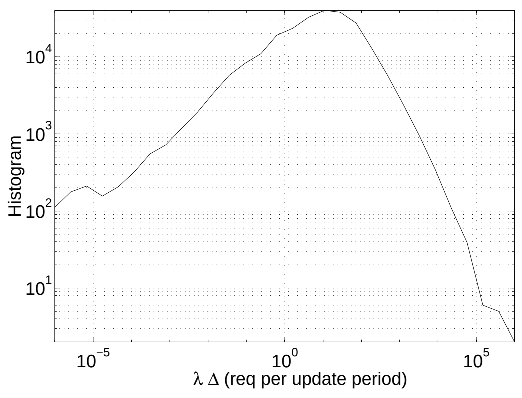
5.5 Trace-driven Simulation

Based on the switching thresholds calculated in Section 5.4.4, we now perform a trace driven simulation to get a feel for how SPREAD will behave in a real-life setting. To that end, we analyze log traces from one access node (POP) at AT&T Worldnet (Bridgeton) over a period of 10 days, collected in May 1999. The total number of requests in the trace is roughly 10 million. From the logs we extract all the cacheable requests that contain last-modified information. We then extract objects of type text/html and image/gif to study how the control algorithm we use in SPREAD will perform. These two object types constitute an overwhelming majority (over 90%) of the accesses. For every single object in the log-file we estimate the average request rate and the average update period. To calculate the average update period we use the average time difference between every request for the same object and the last-modified-time, which is the average update period in the case of exponentially distributed update periods, and is equal to the half of the average update period for periodic updates. In reality, a SPREAD proxy would continuously monitor the request and update rates; however, using the average update period during the trace was a suitable approximation for the purpose of our simulation study.

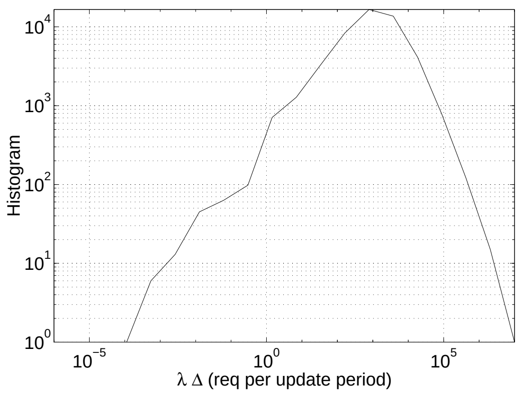
In Figure 5.10(a) we show the distribution of objects of type text/html that have a certain number of requests per update period. We see that most objects have a value which is concentrated between 10^{-4} requests per update period and 10^{4} requests per update period.

Combining the results presented in Figure 5.10(a) and Table 5.1, we can calculate the percentage of objects that would use validation, invalidation, or replication to minimize the bandwidth usage. Table 5.2 shows the percentage of objects requiring every scheme in the case of periodic updates, and the value of the switching points in terms of requests per update period (Th_{VI} and Th_{IR}) for a sample HTML document of size 10KB.

From Table 5.2, we see that in the case of periodic updates, 19% of the HTML objects would require invalidation to minimize bandwidth usage, and 63% would require replication. In the case of exponentially distributed update periods we also calculated the percentage of objects that would require each scheme, and that the percentage of HTML objects that would



(a) Distribution of text/html.



(b) Distribution of image/gif.

Figure 5.10: Distribution of requests per update period $\lambda \cdot \Delta$.

Perspective	Threshold (req per update period)	V	I	R
BW (Periodic)	Th_{VI}=0.7, Th_{IR}=3.6	18.4%	19%	62.2%
BW (Exponential)	Th_{VI}=0.55, Th_{IR}=29	16%	52%	32%

Table 5.2: Percentage of HTML objects that require validation (V), invalidation (I), and replication (R). Periodic and Exponentially distributed Updates

require invalidation increases to 86%.

In the Optimization section, we calculated the bandwidth usage in a proxy's link for a single document with varying requests per update period $\lambda \cdot \Delta$, and the average client's latency for a simple two-tier cache hierarchy. Next, we calculate the total average bandwidth usage and the expected latency for validation, invalidation, replication, and SPREAD. We sum up the bandwidth used by *all* objects, and scale the bandwidth we obtain per update period - to per second - by dividing the result by the object's update period Δ.

Table 5.3 summarizes the results for bandwidth usage, and the corresponding client latency.

From Table 5.3 we see that the bandwidth needed to deliver all documents with validation is quite small since most of the documents in the trace have few requests per update period.

Perspective	V	I	R	SPREAD
Bandwidth (KB/sec)	1.6	5.3	803	1.4
Client Latency (sec)	0.58	0.28	0.11	0.26

Table 5.3: Bandwidth consumption and resulting Latency for validation (V), invalidation (I), replication (R), and SPREAD for HTML documents. Periodic Updates.

Invalidation, on the other hand, has a higher bandwidth usage than validation, since invalidation messages are sent for documents that are never requested. Replication has the highest bandwidth usage since all documents are being replicated, and many are not requested. SPREAD, has the minimum bandwidth usage since proxies automatically select validation, invalidation, or replication to optimize bandwidth. The benefits in terms of bandwidth of SPREAD compared to validation are not very high since there are not many hot documents in the trace that produce a large number of IMS requests (the bandwidth usage of validation would be much higher in the case of more popular documents). However, the latency experienced by the clients with SPREAD is about half the latency experienced with validation. Even though SPREAD is not optimized to minimize latency we see that the latency offered by SPREAD is smaller than for validation or invalidation. As SPREAD proxies subscribe to invalidation or replication to minimize bandwidth usage, the latency reduces, since the origin server is not contacted so often. Of course, replication has the lowest latency at the cost of high bandwidth usage. We have also calculated the same parameters than in Table 5.3 for the case of exponentially update periods, and the results for exponentially update periods do not differ much from those for periodic updates.

Next, we also study the case for objects of type image/gif (see Figure 5.10(b)). GIF objects tend to change less frequently, and therefore the number of requests that a GIF object receives before it is updated is much higher than for HTML objects (Figure 5.10(a)). Table 5.4 shows the total bandwidth usage by GIF objects using validation, invalidation, replication, and SPREAD. From Table 5.4 we see that validation performs worse than it does for HTML documents, since validation results in a higher number of IMS queries to the

Perspective	V	I	R	SPREAD
Bandwidth (KB/sec)	7.6	2.4	72	1.6
Client Latency (sec)	0.45	0.14	0.11	0.12

Table 5.4: Bandwidth consumption and resulting Latency for validation (V), invalidation (I), replication (R), and SPREAD for GIF images. Periodic Updates.

origin server (since GIFs see more requests per update). Replication performs better than it does for HTML documents for the same reason. This also causes SPREAD to improve on invalidation much more than it did with HTML. As before, we see that though SPREAD is tuned to optimize bandwidth, it has an average latency which is very close to that achieved with replication.

Perspective	V	I	R	SPREAD
Bandwidth (KB/sec)	9.2	7.7	875	3
Client Latency (sec)	0.49	0.18	0.11	0.16

Table 5.5: Bandwidth consumption and resulting Latency for validation for validation (V), invalidation (I), replication (R), and SPREAD for HTML documents and GIF images. Periodic Updates.

Finally, in Table 5.5 we add the total bandwidth usage and calculate the average latency for text/html and image/gif objects to see how the various schemes perform. We see that the bandwidth savings and the reduction in latency for SPREAD compared to validation, invalidation, and replication are much more relevant than for either text/html or for image/gif objects alone. That is, while one of the mechanisms may be suited for one type of object, e.g. validation to reduce bandwidth usage for text/html or invalidation for image/gif, SPREAD does well overall, distancing itself from the other mechanisms when a mix of objects are considered.

5.6 Conclusions

In this chapter we introduced SPREAD, a new architecture for content distribution. SPREAD uses a network of proxies that automatically configure themselves and make autonomous decisions on how to maintain cache consistency. They dynamically choose between client validation, server invalidation and replication to optimize bandwidth usage. One key component of SPREAD is that it uses a new class of Transparent proxies called Translucent proxies. Translucent proxies can be cascaded and networked together transparently, without requiring them to be placed at focal points in the network.

SPREAD is also showing promise as a base platform for a large set of other wide-area applications for which self-organization, scalability and robustness are important. An interesting area to pursue the use of SPREAD for reliable multicast, and for broadcasting content.

Chapter 6

Parallel-Access for Replicated Content

Popular content is frequently replicated in multiple sites/caches in an effort to share the load and reduce clients' retrieval latencies. However, choosing the best site/cache is a non-trivial task and a bad choice may give poor performance. We propose a scheme in which clients access multiple sites in parallel to speedup document downloads while eliminating the problem of server selection. In our scheme, clients connect to sites using unicast TCP connections and dynamically request different pieces of a document from different sites. The amount of data retrieved from a particular site varies depending on the network path/server conditions. Dynamic parallel-access can be easily implemented in the current Internet and does not require any modifications at the sites. Using dynamic parallel-access, all clients experience dramatic speedups in downloading documents, and the load is shared among servers without the need for a server selection mechanism. Even in a situation where clients are connected through modem lines, dynamic parallel-access offers transmission rates at least as high as the fastest server.

6.1 Introduction

In order to alleviate the problems of congestion in the Internet, multiple copies of popular documents are often stored in several locations. With network caching, geographically dispersed caches pull and store copies of a document. With mirror site replication, documents are replicated at secondary sites in an effort to both distribute the load of requests across servers and to decrease clients' retrieval latencies. In the rest of the chapter we will refer to caches and mirror servers/sites interchangeably, since both mechanisms are used to replicate popular content in multiple network locations.

When a copy of the same document is placed at multiple sites, choosing the best site is not trivial and the obtained performance can dramatically vary depending on the selected site [40] [27] [81]. However, the fact that there are several copies of the same document in geographically dispersed servers allows clients to access several servers in parallel and obtain from every server a different portion of the document. Using a parallel-access eliminates the need for a complex selection process and performs load balancing among the different servers. Additionally, a parallel access can significantly speedup the transfer of a document. Clients experience a transfer rate close to the sum of the individual transfer rates of the servers contacted.

In this chapter we develop a parallel-access scheme that uses application-level negotiations to schedule the transmission of different document parts from servers. We consider two different parallel-access schemes, (i) **history-based TCP** parallel-access , and (ii) **dynamic TCP** parallel-access . With a history-based parallel-access , clients specify *a-priori* which part of a document must be delivered from each server, e.g., server one sends the first half of the document, server two sends the second half, etc. The size of the part delivered by one server is proportional to its rate, thus, a slow server will deliver a small part of the document while a fast server will deliver a big part of the document. To achieve the maximum possible speedup, all servers must finish transmitting their part at the same time, i.e., *all* servers must be delivering useful data to the client until the document is fully received. To calculate the size of every part, a history based parallel-access uses a database of previous server rates, which is refreshed periodically, e.g. every 10 minutes. We find that a history-based parallel-access scheme can speedup the transmission of a document when the network/server conditions do not change, since it is easy to predict the rates from the client to every server based on previous measured rates. However, in the case where the network/server conditions change rapidly, especially during day time, a history-based parallel-access scheme has poor performance since it is not able to correctly predict the rate of the different servers during the transmission of the document.

Dynamic parallel-access works as follows. A client partitions a document into small blocks and first requests one different block from each server. When a server finishes the transmission of one block, the client requests from this server another block that has not yet been requested from any other server. When the client receives all blocks it resembles them and reconstructs the whole document. Negotiations between the client and the servers indicating which block to get, are performed at the application-level using the HTTP1.1 byte-range header [41]. Multiple application-level negotiations for the same document and the same server server, use the same TCP persistent connection to avoid multiple slow-start phases [70]. For every negotiation between the client and each server, there is a round-trip-time (RTT), during which no data is transmitted (see Figure 6.1). To avoid *idle times*, requests for several blocks to the same server can be pipelined. Thus, before one server ends the transmission of one block, the client requests another block from the same server. The scheme implicitly adapts to changing network and server load. When the number of blocks is large, the degree of granularity is high and it is easy for all servers to deliver useful information until the complete reception of the document.

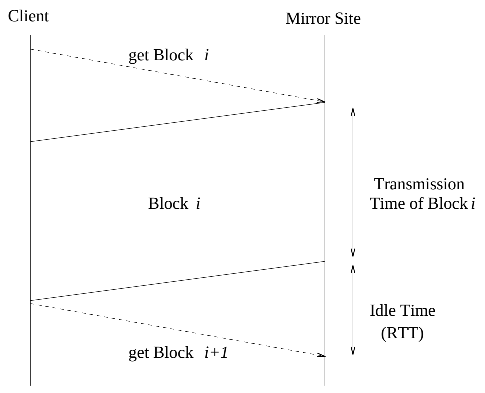


Figure 6.1: Dynamic parallel-access : Block request.

We have implemented a prototype of a dynamic parallel-access scheme as a JAVA client that receives the URL of the mirror servers as input parameters. We evaluated the dynamic parallel-access scheme for a different number of mirror sites, different document sizes and various network/server conditions. Extensive experiments using the JAVA client implementation showed that dynamic parallel-access offers dramatic speedups in downloading a document, even when network/server conditions change rapidly. Very high speedups are obtained when all the servers contacted have similar performance. In the case when one of the servers is much faster than the others, the resulting speedup is not as significant when compared to the fastest server's performance. Even in this latter case, however, the parallel-access scheme offers response times that are at least as low as the ones provided by the fastest server contacted, while avoiding the complicated task of making a server selection. One important advantage of a dynamic parallel-access is that it increases resilience against server failure. If some servers go down while the client is downloading a certain document, a dynamic parallel-access automatically re-assigns the missing blocks to the servers that are still alive and finishes the reception of the whole document.

A parallel-access scheme works efficiently in the case when there is no common bottleneck in the path from the client to the origin server. However, in the case when clients access information through a slow link, e.g. a modem link, connecting to several servers in parallel may not result in an additional speedup. For clients connecting through a modem link, a dynamic parallel-access provides transfer times that are as good as the ones offered by the fastest server. In the case when the mirror servers are very slow, the modem link is not a bottleneck and a dynamic parallel-access reduces the transfer time even more than the transfer time offered by the fastest server.

A dynamic parallel-access to multiple sites provides better speedups than a multiple parallel-connection to a single server, since parallel connections to a single server compete for the same server/network resources. Using a dynamic parallel-access to different mirror servers, parallel connections do not compete among them and the resulting speedup is very high. In addition, with a parallel-access to multiple servers, the number of TCP connections per server is kept low and every TCP connection lasts for a shorter period of time. The load is shared among the servers, and therefore, a higher number of receivers can experience high speedups.

6.1.1 Related Work

Choosing the best mirror site has been subject of research during the last years. Several techniques have been proposed including multicast communication to poll all the mirror sites [15], dynamically probing [27], combining server push with client probes[40], and statistical record-keeping [82]. The work in [82], and the work of others, indicates that the choice of the best server is not always obvious and that the obtained performance can dramatically vary depending on the server selected.

One of the most relevant related work in parallel access is Maxemchuk's work on dispersity routing [66] and Rabin's work on information dispersal [74], where a document is divided into several pieces and each piece also includes some redundant information. The receiver obtains different pieces of the document along different network paths and when the receiver has enough pieces the document is reconstructed. Currently there are several software packages that allow clients to dynamically pause, resume, and jump from one mirror site to another during a document transmission if the current mirror site is very slow [3] [7] [5]. Other software packages allow to open multiple parallel connections to a certain site to speed the download of a certain document [1]. The document is divided into several pieces and different pieces are delivered in different connections.

Byers et al. [21] proposed to access multiple servers in parallel using erasure codes [75] [62]. They proposed a parallel access scheme for open-loop multicast/broadcast distributions. Using erasure codes, servers take the original document, consisting on k packets, and generate h parity packets with the property that *any* k out of the $k+h$ data plus parity packets can be used to reconstruct the original k packets. Servers generate different sets of parity packets and cyclically transmit parities and originals. Clients can recover the whole document as soon as they receive k different packets, regardless of the which server the packets came from [21]. To efficiently encode large documents with small encoding/decoding delays, special erasure codes, such as Tornado Codes[62], must be used. Using Tornado codes, an open-loop parallel-access scheme can be scaled to a large number of clients and servers. However, this approach requires the servers to encode all their documents and the clients to install decoders to reconstruct the encoded documents. In addition some problems still remain unresolved, e.g., how to stop the servers, or congestion control.

In the current Internet, most of the communications are performed via unicast between the clients and the servers. Clients and servers use close-loop communications, exchanging feedback messages at the transport-level (TCP) to implement reliability and congestion control. Thus, we propose to implement a parallel access scheme where clients and servers connect via unicast using TCP. A dynamic parallel-access uses application-level negotiations to dynamically request different pieces of a document from the mirror servers. Using the standard TCP and HTTP protocols, a dynamic parallel-access achieves very good speedups, without requiring to re-encode any document on the server. Our dynamic parallel-access implementation can be easily included in Web browsers without any modification of the mirror servers and no additional buffer requirements at the clients (since current Web browsers already support support opening multiple parallel connections to the same server). It can also be included in cache sharing protocols [37] [80] to speedup the download of popular documents and balance the load among neighbor caches with a document copy.

6.1.2 Assumptions

We consider *popular* documents that are identically (bit-by-bit) replicated on several mirror servers. We consider *large* documents of several hundreds of KBytes. For small documents, several documents should be grouped into a bigger document, before applying parallel-access to the bigger document (i.e. group all images and text from a Web page into a single document).

We assume that the path from the client to the mirror servers is *bottleneck-disjoint*, that is, packets from one mirror server are not slowed down or dropped due to packets from another mirror server. We consider that servers and clients implement the HTTP 1.1 protocol [41] to allow for persistent connections and application-level negotiations.

The rest of the chapter is organized as follows. Section 6.2 presents and analyzes a history based parallel-access under different network/server conditions. In Section 6.3 we present the dynamic parallel-access and demonstrate that it offers dramatic speedups for different document sizes, number of servers, and network conditions. Section 6.4 considers a dynamic parallel-access where a client is connected through a modem link. Section 6.5 compares a dynamic parallel-access with a scheme where the client opens multiple parallel connections to the same server and simulates a dynamic parallel-access with pipelining. Section 6.6 concludes the chapter and discusses some future work.

6.1.3 Mirror Site Discovery

Concerning the discovery of mirror sites, the most frequent approach is to publish a list of mirror sites on the master Web site. Clients, manually select the server that they believe will offer the lowest retrieval time. Some search engines provide a full list of mirror sites and rate them in terms of loss rate and round-trip-time [3]. Several organizations running mirror sites are modifying DNS servers to return to the client the IP address of the administratively closest mirror site [8]. Other recent studies suggest to extend DNS servers [52] or a central directory [43] to return a full list of all servers containing a copy of a certain document. Current cache-sharing protocols [80] [37] keep local information about the location of duplicated document copies in neighbor caches. When a client requests a certain document and the document is not found in the local cache, the local cache will re-direct the request to the best neighbor cache with a document copy.

6.2 History Based Parallel-Access

A history-based parallel-access uses information about the previous transmission rates between the client and every mirror server. It needs this information to decide a-priori which document part should be delivered by each server. The client divides the document into M disjoint blocks, one block for every mirror server. Let μ_i be the transmission rate for server i, $1 \leq i \leq M$. Let S be the document size. Let $\varphi_i S$ be the size of the block delivered by server i and let $T_{t,i} = \frac{\varphi_i S}{\mu_i}$ be the transmission time of this block. To achieve a maximum speedup, all servers must finish transmitting their block at the same time, thus, $T_{t,i} = T_{t,j}$ for all $i, j \in \{1, .., M\}$. When all servers transmit their block at the same time, there are no servers that stop transmitting before the document is fully received. To achieve a maximum speedup the size $\varphi_i S$ of the block sent by server i, must be equal to $\varphi_i S = \frac{\mu_i}{\sum_{j=1}^{M} \mu_j} S$. Fast servers send a bigger portion of the document, while slow servers send smaller portions. The parallel rate μ_p achieved when all servers keep sending useful data until the document is fully received, is equal to the sum of the individual rates to every server $\mu_p = \sum_{i=1}^{M} \mu_i$.

A history-based parallel-access needs to use a database with information about the previous rates from the different servers to the receiver to estimate the rate to every server, μ_i. Instead of having one database per-client, a single database could be shared by a group of receivers connected through a proxy-cache. The database is actualized every time that a client connects to a server or can be actualized periodically with an automated probing from the proxy.

6.2.1 Experimental Setup

To evaluate history-based parallel-access we have implemented a parallel-access JAVA client program that takes as input parameters the URLs and uses a database of previous rates from every mirror server to the client. The JAVA client performs a history-based parallel-access for the requested document, saves the document locally, and records the time it took to download the document. To calculate the size of every block, clients need to know the total document size S. To obtain the document size, the parallel-access JAVA client polls the servers using a HTTP request at the beginning. The document size could also be pre-recorded in a proxy cache or given to the client through a DNS server, thus, avoiding additional RTTs to poll the servers.

To analyze the performance of a history-based parallel-access scheme, we performed several experiments using mirror servers in the Internet. In particular we considered several mirror servers of the Squid Web Page (http://squid.nlanr.net/) [88]. Figure 6.2 shows a network map with the mirror servers considered and the bandwidth of the slowest link in every path as given by pathchar [51]. The Java client is always located at EURECOM, France. Since the servers are situated in different countries and given that the connection from our institution (EURECOM) into the Internet has a high access rate, a parallel-access connection from a EURECOM client to the mirror sites is likely to be bottleneck-disjoint.

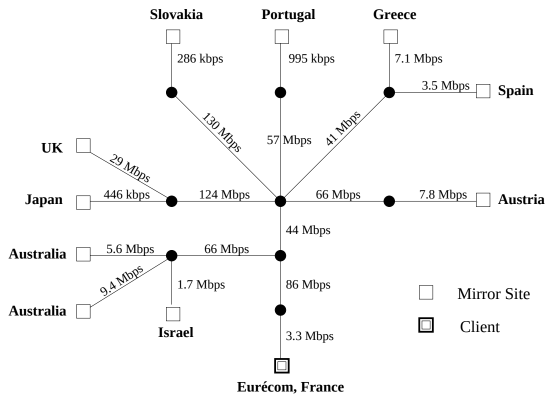


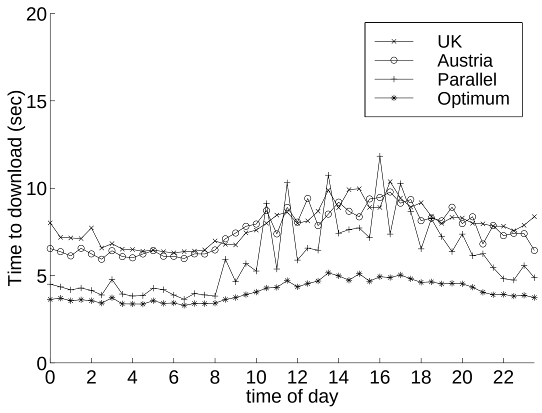
Figure 6.2: Mirror sites for the Squid home page. Client is located at EURECOM, France.

We evaluated a history-based parallel-access scheme every 15 minutes, making sure that different experiments do not overlap. We run the experiments 24 hours a day during a 10-day period and averaged over the 10-day period.

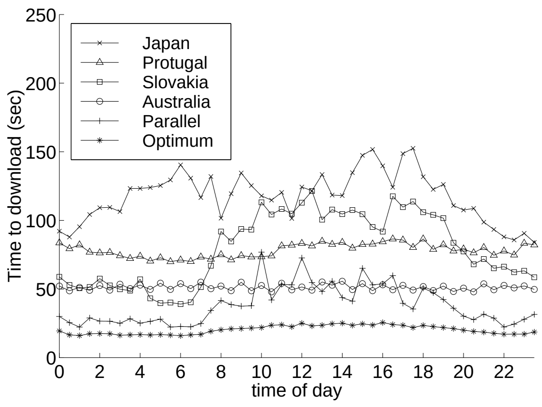
6.2.2 Analysis of the Results

Next, we present the performance of a history-based parallel-access where a client at EURECOM requests a 763 KByte document from two servers (Austria and UK), which have average transmission rates between $80 - 100$ Kbps. The actual document is the beta version of the SQUID 1.2 software [88], gzipped. The database with the previous rates from the client to every server is updated when the JAVA client performs a request for the document, that is every 15 minutes. The client assumes that the average rate offered by every server μ_i, will be equal to the rate obtained 15 minutes before.

In Figure 6.3 we show the transfer time offered by a history-based parallel-access , and the transmission time offered by an individual connection to every server. In addition, we also show the **optimal** transmission time. The optimal transmission time is the transmission time achieved by a parallel-access scheme where all servers send useful information until the document is fully received and there are no idle times between the reception of two consecutive blocks. To calculate the optimal transmission time, the average rates obtained



(a) $S = 763$ KB, $M = 2$.



(b) $S = 763$ KB, $M = 4$.

Figure 6.3: History Based parallel-access .

from every server after the reception of a document are used. We see that during the nights, when network conditions do not vary much, a history-based parallel-access can efficiently estimate the average rate offered by every server and significantly decreases the transmission time compared to the situation where the client accesses a single server. However, during day-time, network conditions rapidly change and estimating the rate to every server using the previous rates achieved, results in bad estimates. Thus, the obtained transmission times with a history-based parallel-access can be higher than the transmission times when clients access every server individually. When the rate to every server is wrongly estimated, some servers stop transmitting before the document is fully received, thus reducing the speedup.

Similar performance of a history-based parallel-access are also obtained for a different set of mirror servers (Figure 6.3(b)). During day times the database could be refreshed more frequently and other more sophisticated estimating algorithms could be used. However, finding the right refresh period and a good algorithm to estimate the rates is not an easy task. In next section, we present another parallel-access scheme that does not use any database and does not need to estimate the rates to the servers. Instead, this parallel-access uses dynamic negotiations between the clients and the servers to adapt to changing network conditions in real-time.

6.3 Dynamic Parallel-Access

We consider a parallel-access scheme that uses dynamic negotiations between the clients and the servers as the transmission of the document progresses. With a dynamic parallel-access the document is divided by the client into B blocks of equal size. To request a certain block from a server, the HTTP byte-range header is used. Clients first request one block from every server. Every time the client has completely received one block from a server, the client requests from this server another block that has not yet been requested from another server. When the client receives all blocks it resembles them and reconstructs the whole document. The following points need to be considered when determining the size of the blocks requested:

- The number of blocks B should be larger than the number M of mirror sites that are accessed in parallel.
- Each block should be small enough to provide fine granularity of striping and ensure that the transfer of the last block requested from each server terminates at about the same time, thus, fully utilizing the server and network resources.
- Each block should also be sufficiently large as to keep the idle times between the transmission of consecutive blocks small compared to the transmission time of a block (see Figure 6.1).

To reconcile the last two points, the document requested via parallel-access should be sufficiently large, i.e. in the order of several hundreds of KBytes.

Since clients need to perform several negotiations with the same server during the transmission of a document, TCP-persistent connections are used between the client and every server to avoid several slow start phases. When there are less than M blocks missing, idle servers may start transmitting in parallel a block that has already been requested from another server but that has not yet been fully received. With this approach, clients experience a transmission rate that is at least equal to the transmission rate of the fastest server. The maximum number of servers that can be transmitting the same block in parallel is limited to two, to avoid a possible bandwidth waste. The bandwidth wasted in the worst case, is equal to $(M-1)\frac{S}{B}$, where $\frac{S}{B}$ is the block size. However, the bandwidth wasted on average is much smaller since slow servers that did not complete the transmission of a block, are stopped after the document is fully received, and only those block-bytes already transmitted by the slow servers are wasted. Decreasing the block size $\frac{S}{B}$, the percentage of a document that may be received duplicated can be made very small. In addition, to avoid any bandwidth waste, clients could easily determine the fastest server during the transmission of the document. Only this server would transmit the missing bytes of the last block and the other servers would be stopped. Thus, slight modifications of a dynamic parallel-access can avoid most if not all the duplicated packets while hardly influencing the obtained speedup.

6.3.1 Analysis of the results

To evaluate the performance of a dynamic parallel-access we implemented the scheme as a JAVA client program. The JAVA program takes as input parameters the URLs of the mirror servers, performs a dynamic parallel-access , saves the document locally, and records the obtained transmission rate. We evaluated the dynamic parallel-access scheme using the experimental setup described in Section 6.2.1. We first consider a dynamic parallel-access to download a 763 KByte document, which is replicated in $M = 4$ mirror sites (Figure 6.4). The actual servers are located in Australia, Japan, Slovakia, and Portugal, to ensure disjoint paths. The average rate to these servers ranges from 5 to 15 KBytes/sec, however, the instantaneous rates greatly fluctuate during the course of the day. We have chosen $B = 30$ blocks. Our current implementation of dynamic parallel-access does not consider pipelining to reduce the idle times (see Section 6.4.1 for a discussion on pipelining).

We compare the transfer time of dynamic parallel-access with the transfer time of an individual access to every server and the optimal transfer time that can be achieved if all servers keep transmitting useful data until the document is fully received and if there are no idle times. From Figure 6.4 we can see that dynamic parallel-access offers very high speedups compared to an individual document transfer from any server. The transfer time is reduced from 50-150 seconds to 20 seconds during all the periods of the day. Even dur-

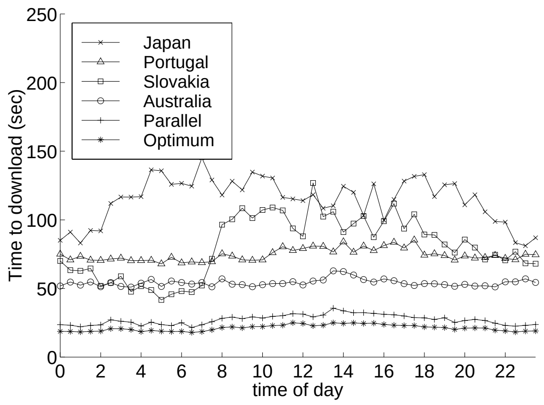


Figure 6.4: Dynamic parallel-access . $S = 763$ Kbytes, $B = 30$, $M = 4$.

ing highly congested periods, where the network conditions rapidly fluctuate, a dynamic parallel-access offers very small transfer times. We observe that the transfer time of a dynamic parallel-access is very close to the optimal transfer time which is an upper bound on the performance of a parallel-access scheme. A dynamic parallel-access is only a couple of seconds slower than the optimum since there are $B = 30$ idle times, which can be avoided using pipelining. Another important advantage of a dynamic parallel-access is that it increases resilience against server failure. In fact during the experiments, there were times were some servers failed while downloading the document. In case of server failure, a dynamic parallel-access automatically re-assigns the missing blocks to the servers that are still alive and finishes the reception of the whole document.

Next, we consider the situation where there are two fast servers (70 KBytes/sec) and two slow ones (10 KBytes/sec). The fast servers are located in Greece and Spain, and the slow ones in Australia and Israel (Figure 6.5). The document size is smaller than in the previous experiment, $S = 256$ KBytes, and therefore we have also reduced the number of blocks to $B = 20$ to avoid that idle times account for a high percentage of the total transfer time (the document is the FAQ from SQUID in postscript format [88]).

We can see that a dynamic parallel-access achieves a transfer time, that is almost half the transfer time of the fast servers (slow servers only contribute sending few blocks and

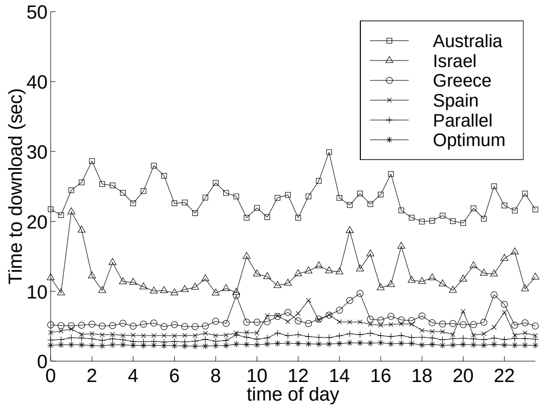


Figure 6.5: Dynamic parallel-access . $S = 256$ Kbytes, $B = 20$, $M = 4$.

decreasing the transfer time of the document by little). The latency benefits may not seem so important if they are compared to the case where a client connects to a fast server (from 4-5 seconds to 2 seconds). However, if the client chooses the wrong server and connects to a slow server, it will end up experiencing transfer times up to 30 seconds.

In the next experiment we consider only two mirror servers (Austria and UK), and perform a dynamic parallel-access for a large document of 5 MBytes (Figure 6.6). Since both servers have a similar rate, a parallel-access will reduce the transfer time by half. The time to download a 5 MBytes document from a single server can be up to 80 seconds. Using a dynamic parallel-access the transfer rate is less than 30 seconds. Due to the high number of blocks used, $B = 60$, the transfer time using the dynamic parallel-access scheme takes a few seconds longer than for the optimal parallel-access . This difference can be avoided by pipelining requests for several blocks (see Section 6.4.1).

6.3.2 Parallel Access for Small Documents

Even though a parallel-access scheme is not intended to be used with small documents, we study the performance of a dynamic parallel-access with small documents of several KBytes.

In Figure 6.7 we see the performance of a dynamic parallel-access scheme for a 10 KB

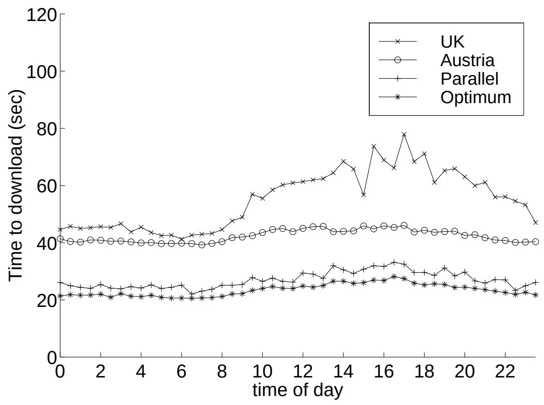


Figure 6.6: Dynamic parallel-access . $S = 5$ Mbytes, $B = 40$, $M = 2$.

document. We considered two mirror servers (Spain and Greece) and $B = 4$ blocks. We see that a dynamic parallel-access has a transmission time close to the transmission time of the fastest server, even though sometimes is slightly higher. Compared to the optimal transmission time, a dynamic parallel-access has a much higher transmission time since the idle times account for a high percentage of the total transmission time. To avoid idle times

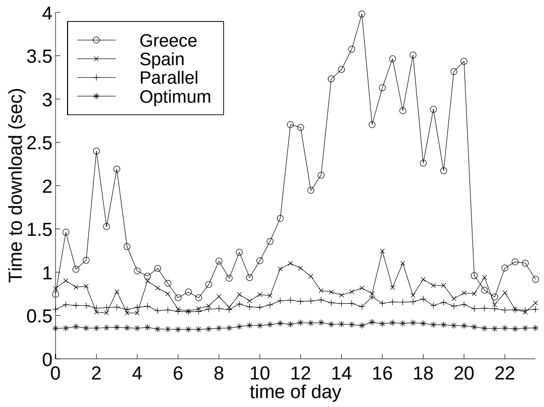


Figure 6.7: Dynamic parallel-access . $S = 10KB$, $B = 4$, $M = 2$.

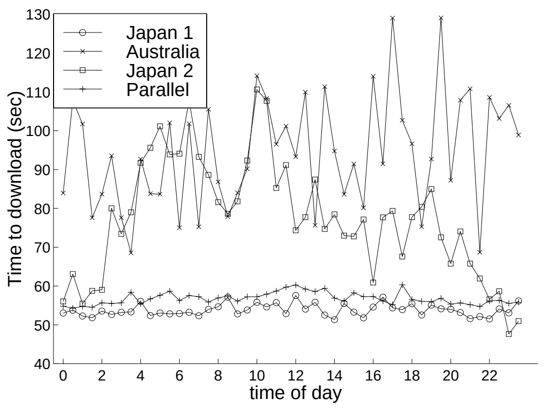
pipelining can be used. However, pipelining requires a minimum block size. The block size should such that $\frac{S}{B} > RTT \cdot \mu$. For instance, if the RTT between the client and most distant server is equal to RTT=100 msec and the server has a transmission rate $\mu = 10$ KBytes/sec, the block size must be $\frac{S}{B} > 1KB$. Thus, if the document size is $S = 10$ KB, and we choose a block size of 2 KBytes, the number of blocks is equal to 5. When the number of blocks is small, the degree of granularity is decreased and it is difficult that all servers keep sending useful information until the full reception of the document without wasting a lot of bandwidth. Thus, for small documents the need for pipelining results in a small number of blocks B, which does not allow to efficiently implement dynamic parallel-access .

In addition, with small documents the connection setup time may account for a significant portion of the total transmission time. A parallel-access scheme speeds up the transmission time of the document but can not do anything about the connection time. If the time to connect to the server is very high, the client may not experience any noticeable difference. To obtain better performances with a parallel-access , several small documents could be grouped together, i.e. all documents in a Web page, and perform a dynamic parallel-access to the bigger document.

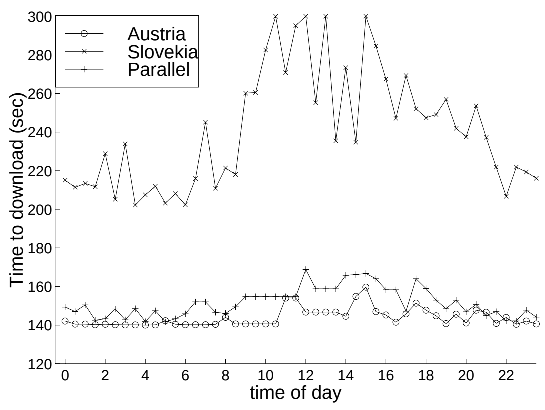
6.4 Dynamic Parallel-Access through a Modem Link

In this section we study the performance of a dynamic parallel-access where a client is connected through a low speed access link, i.e. a modem link. In this case the path from the client to the servers is not bottleneck-disjoint. A single server may consume all the bandwidth of the modem link. Therefore, when a client uses another server in parallel there is no residual network bandwidth and packets from different servers interfere and compete for bandwidth.

In Figure 6.8 we consider dynamic parallel-access for a client connected through a modem line at 56 Kbps. We show the transmission time achieved when connecting to every server individually and when connecting in parallel to all servers using a dynamic parallel-access . In Figure 6.8(a) we considered two slow servers (Japan 1 and Australia) and a fast server (Japan 2). For an individual access to the fast server, the modem bandwidth is fully utilized and the transmission time varies little during all the periods of the day. For an indi-



(a) $S = 256$ KBytes, $B = 20$, $M = 3$.



(b) $S = 763$ KBytes, $B = 30$, $M = 2$.

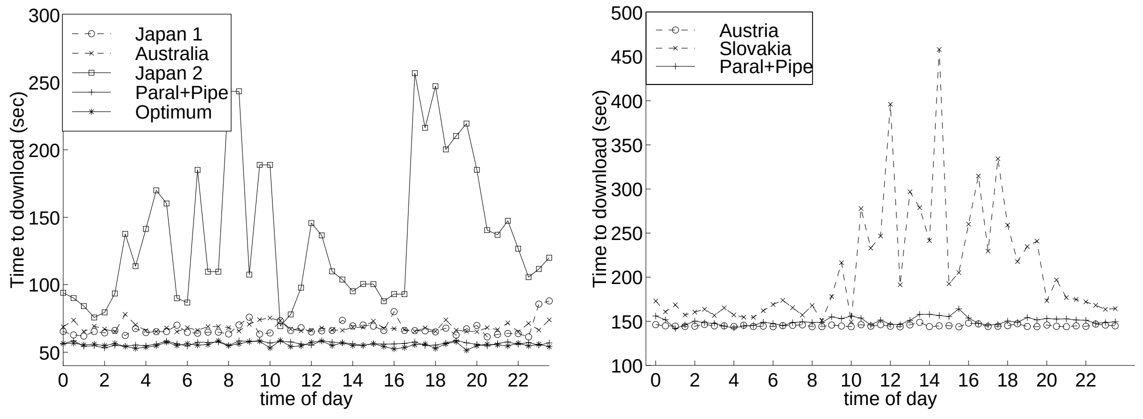
Figure 6.8: Retrieval latency for a parallel-access scheme and for an individual-access scheme to every server using a modem link.

vidual access to the other two slow servers, the rates obtained are about 24 Kbps, which is much lower than the maximum modem link rate (56 Kbps). In this situation the modem link is not fully utilized and the transmission time fluctuates depending on the different levels of congestion in the network/servers along the day. A similar effect can be seen in Figure 6.8(b), where there are only two mirror-servers, a fast one and a slow one.

For the dynamic parallel-access , we see that the transmission rate achieved is close to the transmission rate of the fastest server, which is equal to the transmission rate of the modem link. The fact that with a dynamic parallel-access the transmission rate obtained is always slightly higher than the transmission rate offered by the fastest server is due to the idle times between block requests. Next, we study the performance of a dynamic parallel-access that uses pipelining to avoid idle times.

6.4.1 Simulation of Pipelining

In this section we repeat the previous experiments (Figure 6.8(a) and 6.8(b)) simulating a dynamic parallel-access with pipelining. To estimate the transmission time with pipelining,



(a) $S = 256$ KBytes, $B = 20$, $M = 3$. Pipeline (b) $S = 763$ KBytes, $B = 30$, $M = 2$. Pipeline

Figure 6.9: Retrieval latency for a dynamic parallel-access scheme and for an individual-access scheme to every server through a modem link. Pipelining simulation.

we measured the RTT to every server and then recalculate the transmission time assuming that RTT=0. This simulation gives an upper bound on the performance of pipelining since it assumes that all RTTs can be fully suppressed.

In Figure 6.9 we see that the rate achieved by a parallel-access with pipelining through a modem link. We observe that the rate offered by a parallel-access with pipelining is smaller (Figure 6.9(a)) or equal (Figure 6.9(b)) than the rate achieved by a single connection to the fastest server. In Figure 6.9(a) the transmission rate of each server is much smaller than the maximum modem link rate, thus, a single server connection does not fully utilized the modem link. In this case, a parallel-access with pipelining can speedup the transfer of a document compare to a single server connection, and achieve transmission times that are even smaller than those offered by the fastest server. From Figure 6.9(a) it is also important to notice, that the transfer time achieved with a dynamic parallel-access with pipelining is almost equal to the optimal transmission time. Thus, the additional delay that the JAVA implementation may introduce is very small. The results achieved with pipelining in Figure 6.9(a) can also be applied to the other dynamic parallel-access experiments presented Section 6.3.

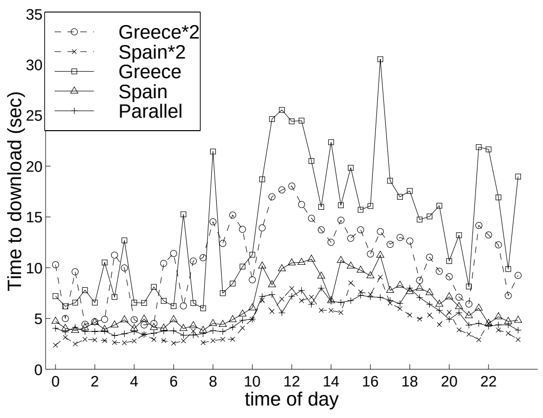
Using pipelining is not so crucial since the performance of a parallel-access without pipelining is already very good, and the expected benefits are small. Implementing pipelining can be easily done and does not require to calculate the exact RTTs but simply estimate an upper bound on the RTTs. Using an overestimated RTT, pipelining could eliminate the idle times with no performance degradation.

6.5 Parallel Access to a Single Server

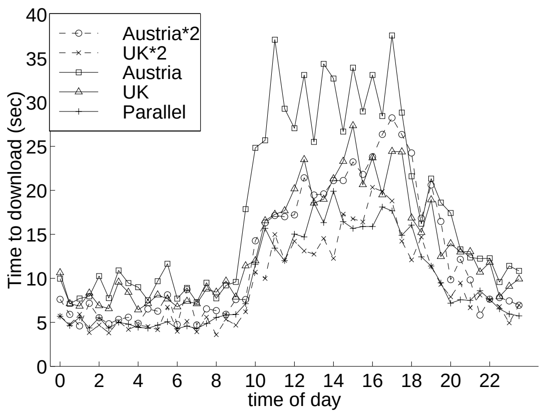
In this section we compare a dynamic parallel-access to multiple mirror-servers with a parallel-access to a single server. For a fair comparison, we consider the situation where a single client opens M TCP-parallel connections to the same server and compare this case to a dynamic parallel-access to M servers. Let μ_s be the rate to the slowest server, and μ_f be the rate to the fastest server. If the residual bandwidth in the path from the client to the server is large enough, a M-parallel connection to a single server with rate μ_i, will have a transmission rate equal to $M \cdot \mu_i$. A dynamic parallel-access to M servers has a transmission rate $\mu_p = \sum_{i=1}^{M} \mu_i$ which is higher than the transmission rate of a M-parallel-access to the slowest server, but smaller than the transmission rate of a M-parallel-access to the fastest server, $M \cdot \mu_s \leq \mu_p \leq M \cdot \mu_f$.

Next, we consider the situation where there are two mirror servers, a slow one in Greece and a fast one in Spain, and perform the following experiments (i) clients retrieve a document with an individual connection to both servers, (ii) clients retrieve a document using a dynamic parallel-access to both servers, (iii) clients retrieve a document using a dynamic parallel-access with two connections to the same server.

Figure 6.10 shows the transmission time obtained for the different schemes and several document sizes. For the fast server in Spain, the available resources from the client to the server are abundant, and therefore a two parallel connections to this server result in a reduction of the transmission time compared to a single connection to the same server. However, when two connections are simultaneously opened to the slow server in Greece, the resulting transmission time is frequently higher than the transmission time obtained if the



(a) $S = 256$ KBytes, $B = 20$.



(b) $S = 763$ KBytes, $B = 30$.

Figure 6.10: Retrieval latency for a dynamic parallel-access scheme to $M = 2$ servers compared to a double parallel connection to the same server.

client would open only one connection to this server. This is due to the fact that both TCP connections compete for the same resources (network bandwidth and server capacity), and packets from one connection may cause loss of drop packets from the other connection. For a parallel-access to both servers, connections use disjoint bottlenecks and do not compete for the same resources. Thus, the transmission time of a dynamic parallel-access connection to both servers is much smaller than the transmission time of a double connection to the slowest server and is very close to the transmission time of a double connection to the fastest server.

In Figure 6.11 we have considered the situation where the client opens four parallel connections to a single server and we compare the obtained speed-up with that of a dynamic parallel-access to both servers. This is not a fair comparison for the dynamic parallel-access to both servers since a client only receives data in parallel from two connections and not four. In the case that the client opens four parallel connections to the fast server in Spain, the transmission time is smaller than a dynamic parallel-access to both servers. However, in the case that the client in France opens four connections with the server in Greece, the transmission time is equal or even higher than a dynamic parallel-access to both servers.

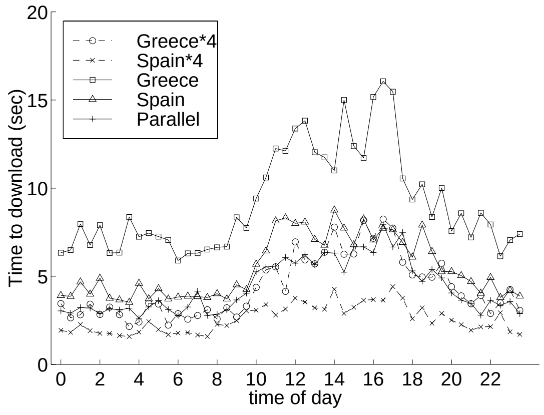


Figure 6.11: Retrieval latency for a dynamic parallel-access scheme to $M = 2$ servers compared to a four parallel connections to the same server. $S = 256$ KBytes, $B = 20$.

Therefore, even though parallel connections to the same server may result in high speedups if the fastest server is selected, it may also result in no speedup if a slow server is selected. On the other hand, a dynamic parallel-access to both servers automatically achieves very good speedups without any server selection. If many clients open parallel connections to the same server, the links close to the server or the actual server may become congested, and clients will not experience any speedup. With a dynamic parallel-access to different servers, the load is shared among the servers and there is a higher number of receivers that can experience significant speedups.

6.6 Conclusions and Future Work

Given that popular documents are often replicated in multiple servers in the network, we suggest that clients connect in parallel to several mirror sites for retrieving a document. We presented a dynamic parallel-access that uses application-level negotiations to speedup document downloads, increase resilience against failure, balance the load among servers, and avoid complex server selections. We evaluated the performance of dynamic parallel-access for different number of servers, document sizes, and network conditions. Dynamic parallel-access achieves significant speedups and has a performance that comes very close to the optimal performance of a parallel-access. Even when clients are connected through modem lines, a dynamic parallel-access offers a transmission time that is close to the transmission time of the fastest server without any server selection. A dynamic parallel-access can be easily implemented in the current Internet and does not require modifications of the content in the mirror sites, in contrast with the digital fountain approach [21].

Future versions of our parallel-access will include pipelining of several blocks to avoid idle times. However, the expected improvement will not very high since a dynamic parallel-access without pipelining already gives transmission times that are very close to the optimal ones. To reduce the number of negotiations between the client and the servers, clients could keep track of the fastest server during the transmission of the first blocks and instead of using a fixed block size, dynamically increase the block size for the fast servers. This approach would require some more complexity at the client to track the fastest servers, but seems a natural extension to our scheme.

Chapter 7

Conclusions

7.1 Summary of the Thesis

The Internet was not designed to distribute large amounts of content to a large number of clients. There is, therefore, a strong need for new and efficient content distribution solutions.

In this thesis we presented several solutions for providing scalable content distribution in the Internet. We focused on i) solutions to efficiently disseminate content close to the clients, ii) solutions to ensure that content delivered to the clients is always up-to-date and synchronized with that at the origin server, and iii) solutions to provide clients with fast access to replicated content and alleviate the problem of server selection.

7.2 Contributions

To evaluate and study content distribution systems, we developed simple theoretical formulations and analytical models. The models developed allow the evaluation of important metrics of a caching system such as hit rate, disk space, client latency, server load, and bandwidth usage, under scenarios including multicasting, and satellite broadcasting. Our research also includes useful empirical results and results from trace-driven simulation that validate our analysis. Our analytical results can be used by Internet Service Providers as well as Content Distribution Networks for an initial specification of the requirements of their content distribution infrastructure.

To efficiently disseminate content close to clients, we first considered caching and multicast. Caching and multicast are two potential techniques for achieving scalable content distribution in the Internet. The efficiency of each technique depends, however, on the type of content being distributed. Our analytical results show that hierarchical caching mimics a multicast distribution, and for most of the content in the Internet results in higher performance. In particular, hierarchical caching provides lower latencies and bandwidth usage than multicast for content that changes not more than once every few minutes. The superior performance of hierarchical caching is primarily due to the fact that caches are located close to the clients, bringing popular content close to the network edge and providing high transmission rates. On the other hand, with multicast the available transmission rate is the lowest rate between the origin server and the clients. As a result, the probability that clients will find a bottleneck and experience frustrating downloading times is high. Hierarchical caching is much easier to deploy than multicast since it does not require any fundamental changes in the network but can be incrementally deployed, and can use standard unicast protocols. For various network topologies we showed that hierarchical caching can mimic the performance of multicast even for a small number of cache tiers (e.g., three or four).

Based on our results about hierarchical caching, we further investigated the advantages of caching hierarchies and their impact in up-to-date content distribution. One popular mechanism to provide up-to-date content from Web caches is to issue an “if-modified-since” poll to the origin server every time that a client request hits the cache [58]. Using analytical results and trace simulations from ISPs and popular content providers, we showed that a caching hierarchy can effectively reduce the traffic inside an ISP and also reduce the number of polls at the content provider’s origin server. For a caching hierarchy to effectively reduce the number of polls, content providers should specify a small period of time (i.e. several minutes) during which the content does not change. In the situation where content providers can not ensure even a small period of time during which the content does not change, the number of polls may overload origin servers. For this latter scenario, we considered a push operation where content is proactively updated at edge caches, thus, removing the need for a poll mechanism. Pushing updates from origin servers directly to a large number of clients or edge caches may result in a non-scalable solution. Using multicast to deliver a high number of document updates may lead to an unscalable multicast architecture since each document requires a new distribution tree. Instead, we presented a push operation that uses a caching infrastructure to reduce the load at each cache and allows for the distribution of a large number of updates to many receivers. To exploit the performances advantages of hierarchical caching and the bandwidth efficiency of multicast, we then presented a new pushing scheme that combines a caching architecture and intra-domain multicast between cache tiers. With caching and multicast push, intermediate caches act as application level filters, only requiring a small and fixed number of static multicast trees, independently of the number of documents transmitted. In addition, the multicast trees are local to an ISP, thus, there is no need for a fully ubiquitous deployment of IP-multicast.

A requirement for a large scale deployment of overlay networks of caches is an automatic mechanism that allows the ad-hoc configuration of such networks with little manual intervention. To easily and automatically deploy application-level infrastructures of caches, we developed SPREAD, an extremely lightweight solution that builds an overlay network of caches on the fly. SPREAD uses a new set of proxies (Translucent proxies) that may be placed in an ad-hoc fashion within the network and spontaneously form a hierarchical content distribution tree. Translucent Proxying Of TCP (TPOT) uses TCP options and IP tunneling to ensure that all IP packets belonging to a TCP connection traverse the proxy that intercepts the first packet of the TCP connection. This allows the ad-hoc deployment of proxy caches anywhere within the network. A TPOT proxy located along the path from the client to the server simply picks up the request and satisfies it from its own cache if it can, otherwise, it lets it pass through. TPOT does not require any extra signaling support or knowledge of neighboring proxies to create distribution trees. In addition to the advantages TPOT proxies offer at the application level, they also have the potential to improve the throughput of intercepted TCP connections at the transport level [77]. SPREAD's network of TPOT proxies automatically reconfigures when proxies go down or come up, or when new ones are added.

A key feature of a scalable content distribution is ensuring strong consistency. Strong consistency may be accomplished by mechanisms such as client validation, server invalidation or replication. Previous proposals have focused on choosing one of these mechanisms in a static fashion. Instead, we considered a dynamic design that automatically selects the best consistency mechanism based on current estimates of the popularity and rate of change of the content, covering the whole range of possibilities. This dynamic approach for ensuring consistency was combined with the SPREAD overlay network described earlier to provide scalability. Proxies within SPREAD autonomously decide on the best mechanism to provide strong consistency for each document, and the selection of a consistency mechanism can be tuned to optimize different parameters, such as bandwidth utilization or client perceived latency. We presented different mechanisms to estimate the request rate and the update rate of each document at different caching levels, and showed that a good estimation can be achieved even with a relatively small number of samples. We successfully tested the performance of our dynamic consistency approach using real logs from a large corporate ISP and showed that it has significantly higher performance in terms of bandwidth and latency than any of the other previous consistency approaches.

To scale the distribution of content to a large number of clients, we also considered using satellite broadcasting combined with a caching system. A point-to-multipoint satellite broadcasting technology can broadcast high quality multimedia content simultaneously to large, geographically dispersed audiences. Unlike terrestrial multicast where information needs to traverse multiple network links and exchange points, a satellite distribution bypasses much of the congested Internet and ensures a high level of quality of service. Once the content has been transmitted trough the satellite, it can be stored in caches at strategic geographical locations and at ISP Points of Presence (PoPs) where users connect to the network. When a user attempts to access a given content, the request is sent to the nearest cache. This model ensures that content is efficiently distributed and stored at the edges of the network, enabling fast client access to rich content. Using analytical models and trace-driven simulations, we evaluated the performance and requirements of a cache-satellite distribution in terms of hit rate, disk space at the caches, document-retrieval latency, and bandwidth needed at the satellite. We found that a cache-satellite distribution that preloads caches with a large number of Web documents can efficiently improve the hit rate of isolated caches with small client populations by 25% to 35%. However, for ISPs with a large client population (e.g., several thousand clients), using a cache-satellite distribution offers only marginal improvements with respect to having an isolated cache. We also showed that caches at the network edge require about 10 TBytes of disk space to store all the content distributed through the satellite during year 1999. The satellite bandwidth required to distribute new content and updates is about 30 Mbps during the same period of time. However, due to the highly skewed distribution of Web documents, with very few documents accounting for most of the requests, simple filtering policies can substantially decrease the requirements of a cache-satellite distribution while still assuring high hit rates. We showed that sending through the satellite those documents that are requested more than once per update period reduces the disk requirements at the caches to 4 TBytes, and the satellite bandwidth to 18 Mbps, while degrading the hit rate by less than 4%.

One way to alleviate the congestion in the Internet is replicating popular content at multiple network locations. With network caching, geographically dispersed caches pull and store copies of a document. With mirror site replication, documents are replicated at secondary sites in an effort to both distribute the load of requests across servers and to decrease clients' retrieval latencies. This wide replication of popular content introduces the possibility of devising new and more efficient client access schemes. We presented and studied the performance of a parallel-access for replicated content where clients access multiple sites in parallel and download different document parts from each site. Determining in advance which document part should be delivered by each server usually provides very inefficient results since network conditions and server load change frequently and are hard to predict. Instead, we proposed a parallel-access where a document is divided into many small blocks. Each server is responsible for delivering only one small block at a time. Once a server is done with one block, it will be asked to deliver another block that has not yet been requested from any server. In this way, fast servers are able to deliver most of the blocks while slow servers still can contribute with some blocks. We showed that a parallel-access performs better for large documents, i.e. several hundreds of KBytes. A parallel-access significantly improves the latency to download a certain document since requests are being parallelized to multiple servers. In addition to speeding up document downloads, a parallel-access offers a set of interesting properties. In particular, a parallel-access increases resilience against failure, balances the load among servers, and relaxes the need for complex server selections. We evaluated the performance of a parallel-access for different number of servers, document sizes, network conditions, and client connections using a real implementation. A parallel-access can be easily implemented in the current Internet without requiring modifications of the origin servers or the content itself.

7.3 Outlook and Open Issues

Internet content is rapidly extending beyond traditional Web content (i.e., text and images) to more sophisticated multimedia content. Examples are streaming audio and video both live and on-demand, high-quality presentations, online education, corporate and training communications, multimedia conferencing, replication of databases, gaming, etc. The Internet was not designed to support the traffic load created by these new applications and requires efficient and intelligent content distribution solutions.

To support scalable content distribution, a multicast-like infrastructure needs to be deployed. Recently several companies have started providing an overlay architecture of nodes/proxies that mimics the performance of multicast by building hierarchical distribution trees [39]. The technology that uses an infrastructure of application-level nodes/proxies to distribute content to a large number of clients is known as *application-level multicast*. Application-level multicast uses unicast connections between network nodes, and does not suffer from the problems that plague IP-multicast. With application-level multicast, content is replicated at every node to avoid that the same content traverses through the same links again and again. Content is then routed through the overlay network using application-level routing costs, and bypassing the traditional IP-routing. An interesting research problem is to study how application-level multicast can efficiently take into account the dynamics of the underlying IP-infrastructure to adapt to changing IP routes, avoiding hot spot generation, and application-level forwarding loops.

Using terrestrial links to deliver multimedia content to large audiences of users with a quality that approaches that of television and radio, is a challenging engineering problem; content needs to traverse multiple terrestrial links and access points that are frequently not administered by a single entity and may be highly congested. On the other hand, a satellite broadcast network can efficiently distribute content at the edges of the network with only one network hop, thus enabling full-screen, full-speed, high-resolution video, stereo sound, and fast and rich text and graphics. A satellite broadcasting infrastructure with caches at the edge of the network will allow new and interesting applications such as pay-per-view, interactive gaming, multimedia conferencing, and targeted advertising. However, due to the broadcasting nature of the satellite, and the limited disk capacity and processing power of local ISP caches, it may be required to implement efficient filtering policies at the ISP side. Using this ISP side filtering policies, ISPs can intelligently preload documents from the satellite that best match the request pattern from local client's, reducing disk space and processing power at the cache. These filtering mechanisms should be integrated with other content management tools to ensure that content is always up-to-date and is delivered to the locations where is mostly needed.

Given the current trend, in a short period of time the Internet will be covered by multiple content distribution networks, forming an *Internet of content distribution networks*. Content distribution networks will evolve and provide more than simple images or text content, targeting other applications such as streaming, e-commerce, dynamic content, or secure transactions. Since different content delivery networks will specialize on different types of content and will offer service to different audiences worldwide, clients should be transparently and automatically redirected to the optimal content distribution network. Thus, there is a need for *content federation* solutions that tight together multiple content distribution networks and provide clients with the optimal content distribution network.

The same way that networks have transport peering relationships to ensure that IP traffic is delivered from one network to another, there will be *content peering* agreements between different content distribution networks. Such agreements will let content providers, content distribution networks, and ISPs build relationships to deliver content to wider audiences and provide a certain quality of service.

The large deployment of content distribution networks will cause the same information to be replicated in multiple network locations. A parallel-access scheme can significantly improve the performance experienced by clients, drastically decreasing the downloading times. The number of replica sites that may contain a copy of a popular document can be very high. However, a parallel-access can offer very good performance advantages even with few replica sites. There is, therefore, a need for an *intelligent replica site scoping* scheme that reduces a large set of replica sites into a smaller set to perform a parallel-access. This smaller set could be formed by those servers that are closer to the client, or those that are in different time zones. The parallel-access scheme presented in this thesis can be easily extended to access real-time streaming content and could be integrated into content distribution networks to improve the performance experienced by clients and efficiently utilize resources.

In the near future, caches and content distribution networks will also be used as content transformers to deliver multimedia information to heterogeneous client devices (e.g. cell phones, pages, PDAs), personalize content to user needs, and provide bandwidth-agile content. The multimedia content distribution market will rapidly grow over the next several years, specially in the wireless Internet area where mobility, thin clients, and new network conditions will bring challenging and interesting problems. The opportunity for content distribution is enormous and gives raise to many open research questions, possible services, and business opportunities.

Bibliography

[1] “Agiletp”, http://www.daemon-info.com/Us/products.htm.

[2] “FreeFlow: How it Works. Akamai, Cambridge, MA, USA. Nov 1999”.

[3] “Go!zilla”, http://www.gizmo.net/.

[4] “Institut EURECOM, Sophia Antipolis, France: http://www.eurecom.fr/”.

[5] “Leechftp”, http://linux.fh-heilbronn.de/ debis/leechftp/.

[6] “National Lab of Applied Network Research (NLANR)”, http://ircache.nlanr.net/.

[7] “Netscape Smart Download”, http://www.netscape.com/.

[8] “World Wide Web Consortium”, http://www.w3c.org/.

[9] K. V. Almeroth, M. H. Ammar, and Z. Fei, “Scalable Delivery of Web Pages Using Cyclic Best-Effort (UDP) Multicast”, In *Proceedings of IEEE INFOCOM*, San Francisco, CA, USA, March 1998.

[10] M. F. Arlitt and C. L. Williamson, “Web ServerWorkload Characterization: The Search for Invariants”, In *Proceedings of the ACM SIGMETRICS*, New York, May23–26 1996.

[11] AYE, “http://www.aye.net/”.

[12] M. Baentsch, L. Baum, G. Molter, S. Rothkugel, and P. Sturm, “World Wide Web Caching: The Application-Level View of the Internet”, *IEEE Communications Magazine*, pp. 170–178, June 1997.

[13] M. Baentsch, G. Molter, and P. Sturm, “Introducing application-level replication and naming into today’s Web”, *Computer Networks and ISDN Systems*, 28:921–930, 1996.

[14] T. Ballardie, P. Francis, and J. Crowcroft, “Core Based Trees (CBT)”, In *Proceedings of SIGCOMM’93*, pp. 85–95, San Francisco, CA, USA, October 1993, ACM.

[15] J. Bernabeu, M. Ammar, and M. Ahamad, “Optimizing a generalized polling protocol for resource finding over a multiple access channel”, *Computer Networks and ISDN Systems*, 1995.

[16] K. Bharat and A. Broder, “A Technique for Measuring the Relative Size and Overlap of Public Web Search Engines”, In *Seventh International WWW Conference*, Brisbane Australia, April 1997.

[17] J.-C. Bolot, S. Lamblot, and A. Simonian, “Design of efficient caching schemes for the World Wide Web”, In *Proc. ITC 15*, Washington, DC, June 1997.

[18] C. Bowman et al., “Harvest: A Scalable, Customizable, Discovery and Access System”, 1996.

[19] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, “On the Implications of Zipf’s Law for Web Caching”, In *Proceedings of IEEE INFOCOM’99*, New York, USA, March 1999.

[20] E. A. Brewer, P. Gauthier, and D. McEvoy, “The Long-Term Viability of Large-Scale Caching”, In *3rd International WWW Caching Workshop*, Manchester, UK, June 1998.

[21] J. Byers, M. Luby, and M. Mitzenmacher, “Accessing Multiple Mirror Sites in Parallel: Using Tornado Codes to Speed Up Downloads”, In *INFOCOM 99*, April 1999.

[22] P. Cao, J. Zhang, and K. Beach, “Active Cache: Caching Dynamic Contents on the Web”, In *Proceedings of IFIP*, pp. 373–388, Middleware, 1998.

[23] A. Chankhunthod, P. Danzig, C. Needaeles, and M. S. K. Worrell, “A hierarchical Internet object cache”, In *Proceedings of 1996 Usenix Technical Conference*.

[24] H. Chen, M. Abrams, T. Johnson, A. Mathur, I. Anwar, and J. Stevenson, “Wormhole Caching with HTTP Push Method for a Satellite-Based Web Content Multicast and Replication Scheme”, In *Proceedings of the 4th International Caching Workshop,* San Diego, March 1999.

[25] K.-I. Chinen, “An Interactive Prefetching Proxy Server for Improvements of WWW Latency.”, In *Proc. INET*, 1997.

[26] D. R. Cox, “Renewal Theory”, 1962, Science Paperbacks series, Methuen‘s Monographs on Applied Probability and Statistics.

[27] M. Crovella and R. Carter, “Dynamic Server Selection Using Bandwidth Probing in Wide-Area Networks”, In *Proceedings of IEEE INFOCOM*, 1997.

[28] P. Danzig and K. L. Swartz, “Transparent, scaleable, fail-safe Web caching”, Technical report, Network Appliance. Santa Clara, CA, USA, 1999.

[29] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, and L. Wei, “The PIM Architecture for Wide–Area Multicast Routing”, *IEEE/ACM Transactions on Networking,* 4(2):153–162, April 1996.

[30] M. Degermark, B. Nordgren, and S. Pink, “RFC 2507: IP Header Compression”, February 1999, Status: proposed standard.

[31] C. Diot, B. N. Levine, B. Lyles, H. Kassem, and D. Balensiefen, “Deployment Issues for the IP Multicast Service and Architecture”, *IEEE Network magazine special issue on Multicasting*, 14(1):78–88, January/February 2000.

[32] M. Doar and I. Leslie, “How Bad is Naïve Multicast Routing”, In *Proceedings of IEEE INFOCOM’93*, volume 1, pp. 82–89, 1993.

[33] M. B. Doar, “A Better Model for Generating Test Networks”, In *Proceedings of IEEE Global Internet*, pp. 86–93, London, UK, November 1996, IEEE.

[34] F. Douglis, A. Feldmann, B. Krishnamurthy, and J.Mogul, "Rate of change and other metrics: A live study of the World Wide Web", In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, December 1997.

[35] B. M. Duska, D. Marwood, and J. Feeley, "The Measured Access Characteristics of World-Wide-Web Client Proxy Caches", In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, December 1997.

[36] H. Eriksson, "MBONE: The Multicast Backbone", *Communications of the ACM*, 37(8):54–60, August 1994.

[37] L. Fan, P. Cao, J. Almeida, and A. Broder, "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol", pp. 254–265, Feb 1998, SIGCOMM'98.

[38] L. Fan, P. Cao, and Q. Jacobson, "Web Prefetching Between Low-Bandwidth Clients and Proxies: Potential and Performance", In *3rd International WWW Caching Workshop*, June 1998.

[39] "FastForward Networks", http://www.ffnet.com.

[40] 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 *Proceedings of IEEE INFOCOM*, San Francisco, CA, USA, March 1998.

[41] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee, et al., "RFC 2068: Hypertext Transfer Protocol — HTTP/1.1", January 1997.

[42] Footprint Overview, "Sandpiper, Thousand Oaks, CA, USA. Oct 1999".

[43] S. Gadde, M. Rabinovich, and J. Chase, "Reduce, Reuse, Recycle: An Approach to Building Large Internet Caches", In *The Sixth Workshop on Hot Topics in Operating Systems (HotOS-VI)*, May 1997.

[44] A. Gove, "Stream on: RealNetwork isn't about to make multicast video a mass medium", *The Red Herring*, November 1997.

[45] S. Gribble and E. Brewer, "System Design Issues for Internet Middleware Services: Deductions from a Large Client Trace", In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, December 1997.

[46] M. L. Gullickson, A. L. Chervenak, and E. W. Zegura, "Using experience to Guide Web Server Selection", , Georgia Institute of Technology, 1998.

[47] J. Gwertzman, "Autonomous Replication in Wide-Area Internetworks", M.S. Thesis, Harvard, Cambridge, MA, April 1995.

[48] J. Gwertzman and M. Seltzer, "World-Wide Web Cache Consistency", In *Proc. 1996 USENIX Technical Conference*, San Diego, CA, January 1996.

[49] X.-Y. Hu, P. Rodriguez, and E. W. Biersack, "Performance Study of Satellite-Linked Web Caches and Filtering Policies", In *Networking 2000*, pp. 580–595, Paris, May 2000.

[50] V. Jacobson, "RFC 1144: Compressing TCP/IP headers for low-speed serial links", February 1990, Status: PROPOSED STANDARD.

[51] V. Jacobson, "Pathchar", http://www.caida.org/Pathchar/ Source: ftp://ftp.ee.lbl.gov/pathchar.

[52] J. Kangasharju, K. W. Ross, and J. Roberts, "Locating Copies of Objects Using the Domain Name System", In *Proceedings of the 4th International Caching Workshop*, San Diego, March 1999.

[53] L. Kleinrock, *Queuing Systems, Volume I: Theory*, Wiley, 1975.

[54] P. Krisnan, D. Raz, and Y. Shavitt, "Transparent En-Route Caching in WANs", In *Work-in-progress in the 4th International Caching Workshop*, San Diego, March 1999.

[55] S. Kumar et al., "The MASC/BGMP Architecture for Inter-domain Multicast Routing", In *SIGCOMM 98*, September 1998.

[56] V. Kumar, "The MBONE FAQ", Collection of Information about MBONE, January 1997.

[57] S. Lawrence and C. L. Giles, "Accesibility of Information on the Web", In *Nature*, July 1999.

[58] C. Liu and P. Cao, "Maintaining Strong Cache Consistency in the World-Wide Web", In *Proceedings of ICDCS*, May 1997.

[59] T. S. Loon and V. Bharghavan, "Alleviating the latency and bandwidth problems in WWW browsing", In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, December 1997.

[60] A. Loutonen and K. Altis, "World-wide web proxies", 1994.

[61] I. Lovric, "Internet Cache Protocol Extension", October 1998, Internet Draft.

[62] M. Luby et al., "Practical Loss-Resilient Codes", In *STOC*, 1997.

[63] A. Luotonen, *Web proxy servers*, Prentice-Hall, 1998.

[64] M. Makpangou, G. Pierre, C. Khoury, and N. Dorta, "Replicated Directory Service for Weakly Consistent Replicated Caches", In *Proceedings of the ICDCS '99 conference*, Austin, Texas, may.

[65] S. Manley and M. Seltzer, "Web Facts and Fantasy", In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, December 1997.

[66] N. F. Maxemchuk, "Dispersity Routing in Store-and-Forward Networks", PhD Thesis, University of Pennsylvania, 1975.

[67] S. Michel, K. Nguyen, A. Rosenstein, L. Zhang, S. Floyd, and V. Jacobson, "Adaptive Web Caching: towards a new global caching architecture", In *3rd International WWW Caching Workshop*, June 1998.

[68] N. Niclausse, *Modélisation, évaluation de performances et dimensionnement du World Wide Web*, Ph.D. Thesis, Université de Nice Sophia-Antipolis, June 1999.

[69] J. Nonnenmacher and E. W. Biersack, "Performance Modelling of Reliable Multicast Transmission", In *Proc. IEEE INFOCOM'97*, Kobe, Japan, April 1997.

[70] V. N. Padmanabhan and J. Mogul, "Improving HTTP Latency", In *Second World Wide Web Conference '94: Mosaic and the Web*, pp. 995–1005, October 1994.

[71] G. Phillips, S. Shenker, and H. Tangmunarunkit, "Scaling of Multicast Trees: Comments on the Chuang-Sirbu Scaling Law", In *Proc. of ACM SIGCOMM'99*, pp. 41–51, Harvard, Massachusetts, USA, September 1999.

[72] D. Povey and J. Harrison, "A Distributed Internet Cache", In *Proceedings of the 20th Australian Computer Science Conference*, Sydney, Australia, February 1997.

[73] Private Communication, "SkyCache".

[74] M. O. Rabin, "Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance", *Journal of the ACM*, 36(2):335–348, April 1989.

[75] L. Rizzo, "Effective erasure codes for reliable computer communication protocols", *Computer Communication Review*, 27(2):24–36, April 1997.

[76] P. Rodriguez and E. W.Biersack, "Continuous Multicast Push of Web Documents over the Internet", *IEEE Network Magazine*, 12, 2:18–31, Mar-Apr 1998.

[77] P. Rodriguez, S. Sibal, and O. Spatscheck, "TPOT: Translucent Proxying of TCP", In Proc. of the 4th International Caching Workshop, Lisbon. Also as Technical report TR 00.4.1, AT&T Research Labs, 2000.

[78] K. W. Ross, "Distribution of Stored Information in the Web", Online Tutorial, http://www.eurecom.fr/ ross/CacheTutorial/DistTutorial.html.

[79] A. Rousskov, “On Performance of Caching Proxies”, In *ACM SIGMETRICS*, Madison, USA, September 1998.

[80] A. Rousskov and D. Wessels, “Cache Digest”, In *3rd International WWW Caching Workshop*, June 1998.

[81] M. Sayal, Y. Breibart, P. Scheuermann, and R. Vingralek, “Selection Algorithm for Replicated Web Servers”, In *Workshop on Internet Server Performance, SIGMETRICS*, Madison, USA, June 1998.

[82] S. Sesham, M. Stemm, and R. Katz, “SPAND: Shared Passive Network Performance Discovery”, In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, December 1997.

[83] SkyCache, “http://www.skycache.com”.

[84] N. G. Smith, “The UK national Web cache - The state of the art”, *Computer Networks and ISDN Systems*, 28:1407–1414, 1996.

[85] R. Tewari, M. Dahlin, H. M. Vin, and J. S. Kay, “Beyond Hierarchies: Design Considerations for Distributed Caching on the Internet”, In *Proceedings of the ICDCS ’99 conference*, Austin, Texas, May 1999.

[86] V. Valloppillil and K. W. Ross, “Cache Array Routing Protocol v1.1. Internet Draft”, February 1998, http://ds1.internic.net/internet-drafts/draft-vinod-carp-v1-03.txt.

[87] Z. Wang and J. Crowcroft, “Cachemesh: A Distributed Cache System For World Wide Web”, In *2nd International WWW Caching Workshop*, Boulder, Colorado, USA, June 1997.

[88] D. Wessels, “Squid Internet Object Cache: http://www.nlanr.net/Squid/”, 1996.

[89] D. Wessels and K. Claffy, “Internet Cache Protocol Version 2”, March 1997, Internet Draft, http://ds.internic.net/internet-drafts/draft-wessels-icp-v2-00.txt.

[90] K. Worrel, “Invalidation in large scale network object caches”, Master’s Thesis, University of Colorado, Boulder, 1994.

[91] Yeager and McGrath, *Web Server Technology*, Morgan Kaufman Publishers, San Francisco, 1996.

[92] J. Yin, L. Alvisi, M. Dahlin, and C. Lin, “Using Leases to Support Server-Driven Consistency in Large-Scale Systems”, In *18th International Conference on Distributed Computing System*, May 1998.

[93] H. Yu, L. Breslau, and S. Shenker, “A Scalable Web Cache Consistency Architecture”, In *Proceedings of ACM SIGCOMM’99*, Cambridge, Sep 1999.

[94] G. K. Zipf, *Human Behavior and the Principle of Least Effort: An Introduction to Human Ecology*, Addison-Wesley, Reading, MA, 1949.

PABLO RODRIGUEZ

Personal Data

Address:	Institut EURECOM 2229, Route des Cretes; BP 193 F-06904 Sophia Antipolis Cedex, France
Tel:	(+33) 4.93.00.26.73
E-mail:	rodrigue@eurecom.fr
Citizen:	Spain

Education

Sep 97- Sep 00	**Ph.D. in Communication Systems** Swiss Federal Institute of Technology, EPFL. Lausanne, Switzerland. Title: ”Scalable Content Distribution in the Internet” Advisor: Prof. Ernst W. Biersack. Institut EURECOM, France
Oct 96-Aug 97	**Postgraduate School in Communication Systems** Swiss Federal Institute of Technology, EPFL. Lausanne, Switzerland.
Sep 95-Jun 96	**M.S. Thesis** Department of Electronic and Electrical Engineering. King’s College, London. UK. Title : ”Optical Fiber Sensors” Advisor: Prof. Alan J. Rogers
Sep 90-Jun 95	**M.S. and B.S. in Telecommunication Engineering** University of Navarra (UPNA), Pamplona. Spain.

Experience

Sep 97- Sep 00	**Research Assistant**, Institut EURECOM, France Conducted research on scalable content distribution schemes for the Internet. Teaching and mentoring: - Supervised four M.S. thesis and several graduate student projects - Assisted in teaching High-Speed Networking and Internet Protocols courses - Teaching and supervising laboratory sessions: IP addressing, spanning tree algorithms.
Jun 99- Oct 99	**Summer Internship**, AT&T Labs-Research. Florham Park, New Jersey. USA. Research on Layer-4 switching, content delivery, and caching.
Sep 95-Sep 96	**Research Assistant**, Electronic and Electrical Engineering. King’s College, London. UK. Conducted research in optical fiber communications and optical fiber sensors. Work performed in collaboration with BICC Cables, UK.

Other Activities

Reviewer for Infocom, Sigmetrics, Computer Networks and ISDN Systems, and Performance Evaluation

Consulting services in the area of Internet content distribution. USA, 2000

Invited speaker at the ACM/SIGOPS meeting on advanced Web applications, May 1999. Paris

Speaker at the Youth European Researchers Conference. European Commission, Brussels (1998)

Member of the Marie Curie Fellowship Association (1997-present)

IEEE Member (1997-present)

Eurecom's coordinator of the Former Students Association at EPFL (1998-present)

Computer skills

Languages	C, C++, Java, Tcl/Tk, Pascal, Basic, Assembly 68000, Shell programming, Perl
Op. Systems	UNIX, Linux, Windows , MS-DOS
Software	Matlab, Maple, Opnet, Mathcad, Pspice, Labview, AutoCAD
Networking	Squid, TCP/IP, Firewalls, IP-Multicast, SNMP

Languages

Spanish	Mother Tongue
English	Fluent
French	Fluent

List of Relevant Publications

P. Rodriguez, S. Sibal, and O. Spatscheck, “TPOT: Translucent Proxying of TCP”, *In Proc. of the 4th International Caching Workshop, Lisbon. Also as Technical report TR 00.4.1, AT&T Research Labs, 2000.*

P. Rodriguez and S. Sibal, “SPREAD: Scalable Platform for Reliable and Efficient Automated Distribution”, *In Proc. of the 9th World Wide Web Conference, EURECOM, ATT Research Labs, May 2000.*

P. Rodriguez, X.-Y. Hu, and E. W. Biersack, “Performance Study of Satellite-Linked Web Caches and Filtering Policies”, *Networking 2000, Paris, May 2000.*

P. Rodriguez, A. Kirpal, and E. W. Biersack, “Parallel Access for Mirror Sites in the Internet”, *Infocom. Tel-Aviv, Israel, March 2000.*

P. Rodriguez and E. W. Biersack, “Bringing the Web to the Network Edge: Large Caches and Satellite Distribution”, *In MONET. Special issue on Satellite-based information services 2000.* Also *In Proc. ACM/IEEE MobiCom Workshop on Satellite-based Information Services (WOSBIS’98), DALLAS, oct 1998.*

P. Rodriguez, E. W. Biersack, and K. W. Ross, “Automated Delivery of Web Documents Through a Caching Infrastructure”, *In submission 2000.*

P. Rodriguez, C. Spanner, and E. W. Biersack, “Web Caching Architectures: Hierarchical and Distributed Caching”, *In Proceedings of the 4th International Caching Workshop, San Diego, March 1999. Also submitted to Transactions on Networking, EURECOM, May 1999.*

P. Rodriguez, E. Biersack, and K. Ross, “Improving the Latency in the Web: Caching or Multicast?”, *3rd International WWW Caching Workshop, June 1998. Also In Computer Networks and ISDN Systems, pp. 2223–2245, 1998.*

P. Rodriguez and E. W.Biersack, “Continuous Multicast Push of Web Documents over the Internet”, *IEEE Network Magazine, 12, 2:18–31, Mar-Apr 1998.*