STREAMING CONTENT DISTRIBUTION NETWORKS WITH MINIMUM DELIVERY COST

by

Jussara M. Almeida

A dissertation submitted in partial fulfillment of

the requirements for the degree of

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN – MADISON

2003

Abstract

A Content Distribution Network, or CDN, is a system to improve the delivery of content to the end users (or clients) in the Internet, in which popular content may be cached or replicated at a number of servers, placed closer to some of the client populations. The design of a CDN consists of defining: (a) which content should be replicated at each server (server content), (b) the number of servers and where they should be placed in the network (server placement), (c) which server a client's requests should be sent to (server selection) and (d) how the server responses should be routed to the clients (routing). CDNs were originally designed for traditional web files (i.e., HTML, image files). However, given the increase in streaming media (i.e., video and audio) content in the Internet, the development of efficient CDN design methods that take into account the special characteristics of media objects is of great interest. These characteristics include the sustained high bandwidth requirements and the new and complex tradeoffs introduced by multicast delivery.

The main goal of this thesis is to develop methods for designing streaming media CDNs with (near) minimum delivery cost, where the delivery cost includes both the server and the network bandwidth costs. We propose and evaluate a new simple minimum cost caching algorithm for conventional CDNs that use unicast delivery. We also develop cost models to determine the optimal server content, server placement and routing for scalable CDNs that use multicast delivery, and new approximate algorithms for determining the near optimal server placement and routing for scalable CDNs. In order to develop the cost models and to evaluate our solutions, a reasonably small set of system and workload parameters that have primary impact on a CDN delivery cost are

identified. These parameters are varied over a wide range of values in order to cover as much of CDN design space as it is practically feasible. This thesis also provides an extensive analysis of the workloads of two educational media servers to provide data for creating realistic synthetic workloads and to gain insights into efficient CDN design strategies.

Acknowledgements

My most special thanks goes to my advisor, Prof. Mary Vernon. I thank her for the discussions, criticism, for teaching me what I know about performance modeling and about what good and solid research really is. I also thank her for the numerous discussions on how to write a good paper. I am very grateful for all the patience and for believing in me and in my potential even when I did not. Certainly I would not have reached this point without her utmost understanding and help.

I would like to thank Prof. Michael Ferris, Prof. Stephen Wright, Prof. Remzi Arpaci-Dusseau, and Prof. Rajan Suri for serving in my thesis committee and for the questions and comments on how to improve the quality of this dissertation. I specially thank Michael Ferris and Stephen Wright for helping me with the optimization models developed in this thesis.

I am extremely grateful to Prof. Derek Eager, who was a constant presence throughout the development of this thesis. I thank him for the valuable technical discussions and advices. I also thank Mike Litzkow and Prof. Lawrence Rowe for providing the media server logs analyzed in this thesis.

I am also very grateful to Pei Cao, who was my advisor during my first two years here in Madison. Pei was not only my mentor then but also a good friend. I will never forget she brought me food when I was sick, on the days prior to my Qualifier exams. During the summer of 1998, as an intern at IBM Research Labs, I was fortunate to work with Erich Nahum and Srini Seshan.

During all these years in Madison, I met many people and made some very good friends. I am eternally grateful to Isabel, Luciana, Andre, Carla, Elaine, Manoj, Shantala, Mary Lestina, Lee,

Miki, David, Han-Yin, Myung Sook, for the constant support and companionship, even in the darkest days, when I could not stand myself. I also thank Su-Hui Chiang for sharing with me her experiences and encouraging me whenever I reached for her, which happened many times, and Jim Gast for the many technical discussions. I am also in debt with my friends in Brazil, who, despite my absence, were always willing to give a word of support and encouragement, reminding me of my roots and of what I intend to become.

This work would have not been possible without the support of my sponsors. In particular, I would like to thank the scholarship from the Conselho Nacional de Desenvolvimento Científico e Technológico (CNPq), in Brazil and the Lawrence H. Landweber NCR Graduate Fellowship in Distributed Systems.

Finally, and most importantly, I thank my family in Brazil for the love, support and inspiration in every moment in my life. This dissertation is dedicated to them.

Contents

List of Tables

List of Figures

Definition of Acronyms

CDN: Content Distribution Network

AS: Autonomous System

VBR: Variable Bit Rate

CV: Coefficient of Variation

LFU: Least Frequently Used

LRU: Least Recently Used

RBC: Resource Based Caching

CMP: Currently Most Popular

IC: Interval Caching

CMP/IC: Currently Most Popular / Interval Caching

GAMS: General Algebraic Modeling System

Chapter 1

Introduction

A content distribution network is a system that places replicates of popular content in a number of servers at various access points in the network with the goal of improving the performance and/or cost of distributing the content to the end users. Content distribution networks were first proposed for delivery of traditional web documents, i.e., HTML files or images, in the Internet. However, *streaming media* files (e.g., audio and video), which are typically much larger objects that need to be transmitted (or *streamed)* to the clients over a significant period of time, are steadily becoming an increasing fraction of the content transmitted in the Internet. Thus, the design of efficient distribution systems for streaming media content, which should take into account the specific characteristics of these objects, is of great interest. The main goal of this thesis is to explore this largely unexplored research topic and develop methods for designing (near) minimum cost content distribution networks for streaming media.

The remainder of this chapter is organized as follows. Section 1.1 defines the key problems associated with designing content distribution networks (or CDNs). The special characteristics of streaming media content, which set it apart from traditional web content and thus motivate the development of new design methods, are discussed in section 1.2. Previous approaches to addressing design aspects of distribution systems are summarized in section 1.3. The goals and the

main contributions of this thesis are described in sections 1.4 and 1.5, respectively. The organization of the remainder of the thesis is given in section 1.6.

1.1 Content Distribution Networks

Due to the explosive growth of web traffic in the Internet, the efficient distribution of stored content has become a major concern among researchers and in industry. During the 1990s, the focus was on the design and implementation of caching strategies for traditional web content. Considered one of the main techniques to reduce network traffic as well as the latency perceived by the clients, proxy caching, where popular content available at an origin server is independently stored at proxy servers (also called local servers since they are located closer to the clients) has been a very active research area. The key challenge behind the design of caching strategies is how to efficiently determine which content should be replicated at the proxies in order to achieve a predefined goal such as to reduce client latency or the traffic in the Internet backbone.

Recently, the Internet has witnessed the emergence of a number of content distribution companies, such as Akamai [7] and Volera [146], which work directly with the content providers in order to improve the delivery service and ultimately the revenue accrued by the company. The organization and terminology associated with a CDN is illustrated in Figure 1. The CDN consists of an origin server, or simply the origin, and a number of proxy servers. Popular content may be (partially) replicated at the proxy servers, which are located closer to some of the client populations. A proxy server may serve a single university campus, a metropolitan area or a wider region involving several states or countries. We use the term “local” network to represent the network where a proxy server and its clients are located. We point out that a “local” network may span a large region depending on the dispersion of the clients that are served by the proxy server. Similarly, the term “remote” network is used to represent the network that connects the “local”

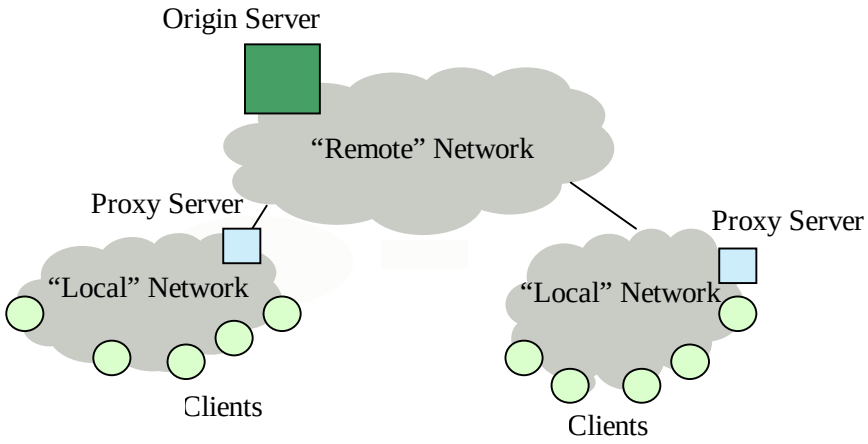


Figure 1: Example of a Content Distribution Network

networks where proxy servers and clients are located to the origin server (e.g., the Internet backbone).

In hierarchical CDN architectures, which are the focus of this thesis, the client sends a request to one of the proxy servers (typically the closest one). If the requested content is not (fully) stored at the proxy, the proxy server forwards the request to the origin server. The origin server either returns the content directly to the client [57, 58], or, more commonly, sends it to the proxy, which then forwards it to the client. A hierarchical CDN may include multiple levels of proxy servers where requests that miss at a lower level proxy are forwarded to a proxy server at a higher level in the hierarchy. The origin server is contacted only if the requested content is not found in any of the proxies. This thesis focuses on a single level hierarchy as shown in Figure 1, which is the most common CDN model. However, we point out that the methods obtained can be extended to CDNs with multiple levels of hierarchy by considering that each proxy at level i is a potential client of a proxy at level i +1 of the hierarchy. A different type of CDN architecture, not addressed in this thesis, is a peer to peer architecture where, when a server receives a request that it cannot (fully)

handle, it searches for another server (any other server) to handle the portion of the request that missed locally.

Key questions behind a CDN design are: a) which content should be replicated or cached at the proxy servers, b) how many replicas and where each replica server should be placed in the network, c) which server a client's requests should be sent to and d) how to route the content from the selected server to the client. Thus, designing a CDN involves the development not only of methods for determining server content (or caching algorithms), but also of methods for determining the placement of servers in the network, server selection and routing. Furthermore, another design problem is to determine the times at which the CDN configuration (e.g., server content and routing) is updated in response to changes in the workload, particularly, in the client request rates. Thus, the design of a CDN should be driven by an understanding of key characteristics of the workload, such as client request rates, object popularity and how these characteristics change over time. These questions are addressed with a predefined performance or cost goal. The goal in this thesis is to design CDNs that minimize the total cost of delivering the content for a given client load, which is related to the price, ultimately paid by the content provider, to have its content delivered to their clients.

1.2 The Special Characteristics of Streaming Media Content

Motivated by the increasing availability of media content in the Internet, improvements of network bandwidth in the Internet backbone and the availability of faster "last mile" connections, such as cable modems and DSL (Digital Subscriber Lines) services, users are becoming increasingly interested in watching movies, listening to radio or music or viewing lectures over the Internet. Consequently, streaming media content (i.e., audio and video) is becoming a significant fraction of the total traffic in the Internet [2, 152].

Streaming media objects and workloads have special characteristics, which distinguish them from traditional web content (i.e., small HTML and image files). These characteristics are: (1) large size, (2) requirement of sustained disk and network bandwidth, (3) possibly unequal access to different portions of the files and (4) several new tradeoffs introduced by the shared delivery of objects if *multicast* delivery is used. The implications of these characteristics on the design of efficient CDNs for streaming media objects are described below.

1.2.1 Media File Size and Bandwidth Requirements

A media file may require tens of megabytes to several gigabytes of storage depending on its quality and duration. This implies that most of the cached content must be stored on disk. Furthermore, storing a new media file on the disk requires a significant amount of disk bandwidth, sustained, typically, over a long period. Hence, not only the disk space, but also disk bandwidth and i/o network bandwidth must be carefully managed when designing a streaming CDN. In particular, it may be important to avoid allocating disk bandwidth to store a requested object in the cache if that object will not be requested again soon. This is particularly true because that bandwidth could otherwise be used to serve client requests for other objects already stored in the cache. Depending on the workload, this may be an important factor to consider in the design of caching strategies for streaming content. In particular, it motivates the development of new algorithms that do not cache objects without prior knowledge of its client access rates. This implies a significant deviation from traditional web caching algorithms where space is the only resource that is managed and that, typically, write a new object into the cache every time a request misses in the cache.

1.2.2 Partial File Accesses

Unlike traditional web clients, streaming media clients may only be interested in receiving a portion of a file, as they browse through content or request portions of the videos that especially interest them. Furthermore, clients with different bandwidth connections to the network (e.g. 56Kbps modem or a 1.5Mbps T1 line) require different quality levels for the streams delivered by the same server. Hierarchical or layered encoding [145] is a technique where each stream is encoded into a base layer that contains the basic information and higher layers that provide increasingly enhanced quality information. Clients with different bandwidth connections can request a different number of layers of a given file. Partial file caching of media objects may also be desirable for performance reasons. By caching only the initial frames of a file (*prefix caching*), a proxy cache can hide latency and loss effects of the link between proxy and origin server [105, 128, 148]. Thus, unlike web CDNs, a streaming CDN design should consider partial file delivery and storage (i.e., file segments or layers).

1.2.3 Tradeoffs for Multicast Delivery

The delivery cost associated with a media object is the cost (in dollars), ultimately paid by the content provider, to have the object delivered, through the network, from the server(s) to the clients. The exact charging schemes for object transmission in the Internet are not well defined, i.e., it is not clear whether the transmission of an object should be charged based on the number of routers or on the number of Autonomous Systems (ASes), which are groups of routers under the same administrative domain, crossed. In either case, the total delivery cost is related to the sustained (server and network) bandwidth required for the transmission. More specifically, the delivery cost for a media object includes the costs of server and network bandwidth required for

transmitting the object to the client. The costs of other system resources, such as processors, disks, network interface cards, are included in the server bandwidth cost since additional resources are needed as the required server bandwidth increases.

For unicast, which is the more common delivery mechanism currently used in the Internet, each client request is served by allocating a new, independent stream and is, thus, charged with the cost of one stream of bandwidth. Thus, for unicast CDNs, total delivery cost is minimized if the number of bytes that are delivered from the client's closest proxy server (i.e., the proxy byte hit ratio) is maximized. In that case, whereas the total server delivery cost is the same, total network delivery cost is minimum. This is true for unicast delivery of media and of traditional web content.

The high bandwidth requirements of streaming media content motivate the use of scalable multicast (or broadcast) delivery, in which, the data follows a delivery tree from a source to all members of a multicast group and packets are replicated only at necessary branches in the tree. For popular objects, significant savings in both server and network bandwidth and, thus, in the per-client and total delivery cost can be achieved if multiple clients share the same multicast stream. Several new scalable streaming protocols that use multicast have been proposed recently [33, 59, 60, 82, 83, 129], motivating the development of efficient scalable CDNs (i.e., CDNs that use multicast) for streaming media.

However, the shared delivery of popular objects introduces several complex new tradeoffs. These tradeoffs are *key* factors to be considered in the design of efficient scalable distribution networks but are not well understood. First, even if each client requests the entire file, earlier portions of a popular file are multicast more frequently and are shared, on average, by a smaller number of clients than later portions. Hence, caching prefix of files, instead of fewer full files, might be more cost-effective. Thus, the possibility of storing prefixes (possibly of arbitrary length)

also needs to be considered in a minimum cost CDN design. Second, there is a key tradeoff between storing the most frequently accessed objects closer to the clients, or storing less frequently accessed objects that have lower cost sharing. As an example, consider a CDN with two proxy servers (proxy 1 and proxy 2) located close to two client populations and assume each server uses a scalable multicast streaming protocol called Bandwidth Skimming [60, 61], which has server bandwidth requirement that is proportional to the logarithm of the client request rate for the full object. Let N_1 and N_2 be measures of the request rate for a given popular object from the two client populations. If each client requests the full object from its local proxy server, proxy 1 needs $\ln(N_1 + 1)$ units of concurrent bandwidth to serve its client population and proxy 2 needs $\ln(N_2 + 1)$ units of concurrent bandwidth to serve its client population. However, if the object is not cached at the proxies, the origin server needs only $\ln(N_1 + N_2 + 1) < \ln(N_1 + 1) + \ln(N_2 + 1)$ units of bandwidth. If N_1 and N_2 are large, total server cost is significantly reduced if the popular object is not cached at the proxies. The savings in server bandwidth costs may outweigh the cost of the remote network bandwidth needed if the content is not stored close to the clients. Thus, unlike for unicast CDNs, maximizing proxy byte hit ratio does not necessarily minimize delivery cost in scalable CDNs.

Furthermore, the optimal content for a given proxy server is interdependent with whether other proxies store that content. In addition to this fact, although the total network cost decreases as the number of proxy servers increase, unlike for unicast CDNs where total server cost is independent of the number of servers, total server cost for scalable CDNs increases with the number of servers. This is because the total number of streams delivered by the servers increases, as illustrated in the example above. Thus, unlike for unicast CDNs, the cost-effectiveness of adding servers, or in other words, the cost-effectiveness of storing content at proxy servers in a scalable CDN is not clear and depends on the relative cost of network and server bandwidth. Therefore,

minimizing delivery cost for scalable CDNs requires knowledge of the relative quantitative cost of a unit of server bandwidth versus the cost of a unit of network bandwidth.

The multicast tradeoffs also need to be considered for server placement, server selection and routing in scalable streaming CDNs. As an example, for unicast delivery, network bandwidth costs are minimized if clients contact the closest server and the content is routed using the shortest path. However, for scalable delivery, it might be more cost-effective to route content from a server to a given new client using a longer path than the shortest path if that (longer) path is (partially) shared among a larger number of clients. This is because the incremental network bandwidth cost due to the delivery of content to the new client over the longer path may be lower than the incremental cost of delivering it over the shortest path. Consequently, the total network bandwidth cost for delivering the content to all clients may be lower if the longer path is used.

Given these new tradeoffs, the design of scalable CDNs is considerably more complex than the design of conventional unicast CDNs. In addition to system and workload parameters, it must also include the server and network costs of local and origin streams, which depend on the server and the network bandwidth that a given configuration implies.

The design of efficient CDNs that handle the increasing traffic of streaming media files is still a challenging open research problem. Previously proposed algorithms for traditional web content do not take into account the special requirements of these files and, thus, may be suboptimal. Furthermore, the design of an efficient CDN must be driven by a good understanding of the parameters and characteristics of the workload. This thesis explores new CDN design models and heuristics that address the unique characteristics of streaming media and evaluates the performance of the new design techniques over a large design space.

1.3 Previous Approaches

The following sections summarize the most relevant previous work related to this thesis.

1.3.1 Server Content

Defining which content to replicate or store at a number of servers has been addressed in the context of proxy caching in a number of previous works. For traditional web caching, previous algorithms attempt to maximize proxy byte hit ratio which will minimize delivery cost for the given proxy server clients. These policies store a new object into the cache at every cache miss, possibly replacing one or more files already in the cache to free space. The objects that are selected for eviction are those that have lower priority in terms of recency or frequency of previous accesses [130] or a cost measure that is a function of both the access recency and/or frequency and the file size [1, 16, 35, 86, 98, 149, 153]. These policies frequently change the cache content, which implies a high disk write overhead if used for media objects. In particular, a file that is requested very infrequently is written to the disk every time it is requested. Furthermore, for CDNs that use multicast delivery, maximizing byte hit ratio alone is not sufficient for minimizing delivery cost, as discussed in the previous section.

The Resource Based Caching (RBC) algorithm for unicast streaming media files [137, 138], the best unicast caching algorithm previously known, has two key features. First, on each cache miss, the policy *evaluates* whether to cache the requested content or not. The decision is based on comparing a cost measure, which is a function of access frequency and size, assigned to the new content, against the same measure for each object currently in the cache and on the current utilizations of disk space and disk bandwidth. The goal of the policy is to keep disk space utilization and disk bandwidth utilization approximately equal. Second, RBC caches a mixture of

full files and intervals. An interval [51] exploits temporal locality in two sequential media requests that occur close in time to store data that is guaranteed to be needed soon in the future at the expense of high bandwidth requirements. One possible drawback of RBC is that it does not allow unused bandwidth allocated to a cached full file to be used to serve a request to a different file. In [138], the authors conclude RBC outperforms the simpler Least Frequently Used (LFU) [130] caching policy, which caches only full files and evicts from the cache the least frequently used file whenever space is needed to store new content. However, due to a restricted design space considered and to incongruous results, their study is not conclusive. Thus, a more complete analysis of this policy is necessary. In particular, extending the algorithm to incorporate the cost tradeoffs of scalable multicast delivery is also an open question. Furthermore, given the high bandwidth requirements, whether disk-based interval caching is an effective technique is still an important question that remains unanswered.

The first attempt to address the complex tradeoffs introduced by multicast delivery was in [57, 58]. The authors propose extensions to the scalable dynamic skyscraper streaming protocol [59] to enable file prefixes to be cached at proxy servers and analytical cost models that are solved to determine the cache content that minimizes delivery cost, which includes both remote and local bandwidth costs, assuming the streams are delivered using the dynamic skyscraper protocol. The models are solved for a set of average file request rates from each proxy client population, assuming bounds on cache space and disk bandwidth and a simple parameter that is the ratio of the average cost of a proxy multicast stream to the average cost of an origin multicast stream. The authors find that the proxy content that minimizes delivery cost includes prefixes in the order of 15% of the full length of the media stream for files with a range of request rates. However, the delivery protocol used allows only a very strict partitioning of the data where either a fixed length

prefix or the full file might be cached at the proxies. Open questions include: (a) how to extend more efficient and flexible protocols (e.g., Bandwidth Skimming [60, 61]) to enable proxy prefix caching, (b) how to modify the cost models to determine the optimal cache content for these protocols, (c) how to incorporate more precise network delivery costs into the model, (d) the optimal solution for server placement, server selection and routing and (e) rules of thumb or effective approximate algorithms for server placement and (proxy) server content for a wide range of CDN configurations and workloads.

A series of more recent studies [76, 118, 143, 144, 147], parallel to this thesis, use similar optimization cost models, modified for other protocols such as the less efficient scalable Patching protocol [33, 82, 129] and unicast delivery from the origin to the proxy servers. Three of these parallel studies [76, 144, 147] assume bounded proxy storage space and unbounded proxy disk bandwidth. The two other studies [118, 143] assume the proxy servers only store file prefixes (rather than full files). The results in chapters 6 and 7 of this thesis, which address scalable CDN design strategies for more efficient protocols, more general and realistic system assumptions and a wider range of workloads, will be compared against the results from these parallel studies.

1.3.2 Server Placement, Server Selection and Routing

Server placement, selection and routing are CDN design aspects that have been previously addressed mainly for unicast delivery of web content. Defining the optimal number and placement of servers in a network has been shown to be a NP Complete problem for general topologies [70]. Hence, previous works in this area proposed either optimal solutions for very specific topologies, such as trees [93, 96] or approximate algorithms for more general topologies [85, 115, 116]. In either case, the algorithms are applied to unicast delivery and do not attempt to optimize number and placement of servers for multicast delivery.

Similarly, server selection and routing has been explored mainly for unicast delivery. Previous strategies select the closest server or use the shortest path, where the path distance is based either on static measures, such as (weighted) hop count [49, 63], or on dynamically collected measures of path delays [13, 14, 36, 71, 110]. To the best of our knowledge, server selection and routing strategies that specifically account for multicast delivery have only been partially addressed very recently, in two parallel studies [67, 155]. These studies propose heuristics for routing assuming (a) a single live multicast stream [67] or (b) a single server with fixed placement and on-demand delivery using bandwidth skimming streaming protocol [155]. We point out that for live multicast streaming, where the per-hop network bandwidth is independent of the number of clients receiving the stream, the optimal routing is known a priori, and consists of using the delivery tree with fewest links. However, the optimal routing for on-demand delivery of stored content, using the efficient bandwidth skimming protocol [60, 61] is not known, as the per-hop network bandwidth for this protocol depends on the number of clients receiving the stream through that hop. Thus, routing strategies for on-demand multicast delivery of stored content, using bandwidth skimming, assuming multiple servers are still an open research area.

1.3.3 Workload Analysis

The characteristics of web workloads have been extensively analyzed in the past [17, 18, 20, 22, 25, 35, 48]. Workloads containing mainly streaming media objects have been only partially analyzed in a number of recent works, some of which are parallel to this thesis [4, 42, 44, 78, 79, 103, 104, 109, 142]. These media workload analyses focus only on a few statistical workload characteristics (e.g., request arrival rates and file access frequencies or only session characteristics such as number and type of interactive requests, etc). A more complete understanding of media workloads is necessary in order to develop efficient design methods and heuristics for streaming

CDNs as well as to drive the development and improvement of several streaming applications (e.g. distance education servers, multi-conferencing software, etc).

1.4 Goals of this Thesis

The main goal of this thesis is to address the open problems in designing streaming media CDNs, i.e., determine the server content, placement and routing that have minimum total or per-proxy delivery cost, where the delivery cost includes both server and network costs. Efficient approximate algorithms that approximately minimize total delivery cost are also of interest.

CDN design methods for server placement, content and routing rely on system configuration and workload parameters that accurately capture the most relevant system characteristics and impact CDN delivery cost. Thus, two important related goals of this thesis are: (a) to define a set of system parameters that can be used in designing minimum delivery cost CDNs and (b) to analyze actual streaming media workloads more completely than previous work. The workload analysis goals include providing a characterization that could be used to generate a synthetic workload and gain insights into the design of efficient CDNs.

For unicast streaming CDNs, previous methods can be used to determine server placement and stream routing. Furthermore, storing the currently most frequently requested files (or file segments) at each proxy results in maximum byte hit ratio and thus, minimizes delivery cost. The key open problems to be studied are: (a) whether a simple algorithm that stores the objects (files or file segments) with highest measured access frequency will achieve near optimal delivery cost or whether the delivery cost can be further significantly reduced if the proxy also stores short intervals of data for the currently active requests and (b) evaluate how delivery cost for a policy that stores the most frequently requested content compares with the best previous caching policy, RBC.

Methods for designing scalable streaming CDNs have only been partially addressed before. Formal models have been proposed to determine the optimal proxy cache content for the dynamic skyscraper protocol [57, 58]. Furthermore, heuristics have been recently proposed for routing of a single live multicast stream [67] and for routing from a single previously placed server to a group of clients [155]. Important open questions that remain to be addressed and are, thus, key goals of this thesis include: (a) extend the more efficient and flexible Bandwidth Skimming streaming protocol to enable proxy caching of file prefixes, (b) create formal models to determine the optimal proxy server content for the new protocols, (c) create CDN design models that consider more precise network delivery costs to compute the optimal server placement, server selection and routing and (d) develop new efficient heuristic algorithms for placement and routing and evaluate them using realistic Internet topologies and the optimal solutions as benchmark.

We point out that the focus of this thesis is on developing methods for designing a minimum cost CDN for given system and workload parameter values. The CDN solutions obtained with our methods should be recomputed periodically to reflect changes in the system or workload (e.g, client request rates, file access frequency). This is true for the design of other CDNs as well, in particular for the design of traditional web CDNs. The frequency at which the CDN should be updated as the system configuration or the workload changes is not addressed in this thesis, although our analysis of the two media server workloads provides some useful initial insights regarding this issue. Similarly, methods for measuring and detecting changes in the workload are also not addressed in this thesis. These topics are left for future work.

1.5 Main Contributions

The main contributions of this thesis are:

1. We identify a reasonably small set of system and workload parameters that can be varied in order to explore a large region of the CDN design space. Unlike previous work, which uses absolute parameters, we make use of normalized parameters, which capture characteristics of the system and workload that independently influence optimal CDN design. Consequently, we are able to evaluate our proposed CDN design methods over a significantly larger design space than previous work.
2. We perform a much more complete analysis of the workload of two educational streaming media servers than previous and parallel studies. New aspects of the workload characterization that provide useful insights for CDN design include: a) request and session rate as a function of time, b) distribution of request and session inter-arrival times during periods of stable arrival rate, c) insights into how relative file popularity changes from one day to another and from one hour to the next, d) distribution of file access frequency during each day, e) distribution of file segment access frequency as a function of file popularity and f) expected fraction of all cache misses that are to infrequently accessed files.
3. We find that a significant fraction of the new content accessed each hour is not accessed again for one, two or even eight hours in the future, which implies that storing content with no prior knowledge of its access rate may result in significant wasted disk bandwidth. Furthermore, for moderately to highly popular files, the frequency of accesses to each ten-second file segment is approximately uniform, despite a large fraction of requests to very small file segments (high client interactivity). For the least popular files, the frequency of accesses is skewed towards earlier segments. This implies that it may not be necessary to measure access frequency for each file segment to decide which segment to cache. In fact, one can measure the frequency to only a few segments and then fit a curve to estimate the skew of the distribution, if there is one.

Furthermore, for unicast CDNs, full file caching of the most popular files might be the best strategy. We also find that, despite the high client interactivity, up to 30% of server bandwidth can be saved if multicast delivery at is used at one of the servers.

4. Our evaluation of alternative caching algorithms for unicast streaming CDNs shows that, in practical terms, the new and much simpler Currently Most Popular policy caching, which stores at the proxy only the files with the highest measured request rates, outperforms the RBC policy. To further analyze interval caching, we propose and evaluate two new caching algorithms for unicast CDNs: Pooled RBC and CMP/IC. Pooled RBC is based on a key improvement to RBC, which is the sharing of allocated proxy server bandwidth among multiple cached files. CMP/IC is a hybrid policy that uses the simple CMP policy plus a reserved portion of memory or disk storage for caching intervals. Comparing CMP, CMP/IC and Pooled RBC, we find that the simplest policy, CMP, delivers very competitive performance, compared with the more complex Pooled RBC, and CMP/IC. Thus, the benefit of interval caching for unicast streaming is at best modest. Furthermore, since the expected performance benefit of interval caching is even more limited for scalable streaming systems, interval caching need not be considered in scalable CDN design.

5. We propose three new modifications to the scalable multicast bandwidth skimming protocols that enable prefix caching at proxy servers. For each modification, we develop a new optimization model for deriving the minimum cost CDN design. The optimization models are applied to gain insights into the cost-effectiveness of storing content at proxy servers and into the optimal proxy server content for a large region of the CDN design space. These evaluations include an assessment of the effectiveness of alternative protocol optimizations, including: (a) batching (where client requests are grouped and served with a single stream) versus non-

batching of client requests for suffix streams from the origin server, (b) multicast versus unicast delivery from the origin server and (c) the impact of the available client bandwidth on the optimized CDN delivery cost.

6. We determine that storing content at proxy servers that use multicast streaming is cost-effective only if: (a) the origin server uses unicast, or (b) the file request rate is very low, in which case multicast is not very effective or (c) the ratio of the average cost of a proxy stream to the average cost of an origin stream is roughly 1/*P*, where *P* is the number of proxy and the cost of either stream includes both the server and network bandwidth costs.

7. For scalable CDNs, the optimal proxy content, over a large region of the design space, consists of full files rather than file prefixes. We show these results are consistent with previous and parallel observations. Furthermore, in this case, the simple bandwidth skimming protocol (without batching) performed at proxy and origin servers is the optimal delivery protocol. For a limited region of the design space, defined by small per-file request rate, large number of proxies and origin multicast streaming, prefix caching and batching of clients for origin streams significantly reduce delivery cost.

8. We develop a new optimization cost model to determine the number and placement of servers and set of delivery trees from the servers that minimize the total cost of delivering an object to a given set of client sites. We apply this model to show that, for practical and realistic Internet examples, using the placement and routing that are optimized for unicast delivery to scalable multicast CDNs can result in significant increase in the delivery cost compared to the optimal.

9. Using simple canonical topologies we show that, unlike for unicast CDNs, simple rules of thumb such as placement of servers near the client sites that have highest demand [115] or at the most connected nodes [85], do not perform well for scalable delivery. We also show that

the maximum cost increase associated with the use of shortest path routing, the basis for routing in today's Internet, for scalable delivery, though bounded, can be significant.

10. Based on insights drawn from the analysis of simple canonical topologies, we develop six approximate heuristic algorithms for server placement and routing, which require significantly less execution time than the optimization models, and thus can be applied to large problems. Our best approximate algorithm is reasonably accurate, yielding systems with delivery cost within 16% of optimality for a number of Internet examples tested.

1.6 Organization of this Thesis

The organization of the remainder of this thesis is as follows. Chapter 2 summarizes previously proposed strategies for caching, server placement, selection and routing, main findings from previous web and streaming media workload characterizations and a brief description of multicast streaming protocols, for a better understanding of the CDN design models developed in this thesis. Chapter 3 describes the main system design considerations including a description of the delivery cost functions developed for scalable CDNs, the overall evaluation methodology used and the main system and workload assumptions and parameters.

Chapter 4 presents the results of the analysis of the two media server workloads. The performance of the previously proposed RBC caching algorithm is reevaluated in chapter 5. This chapter also includes the description and evaluation of the three new policies, Pooled RBC, CMP and CMP/IC as well as a discussion on the effectiveness of interval caching. Optimal server content for scalable CDNs is the focus of chapter 6, which defines the three new CDN protocols and corresponding cost models and provides results on the optimal delivery protocol and proxy

server content for scalable CDNs. It also identifies the region of the design space where it is cost-effective to store content at proxies that use multicast.

Server placement, selection and routing for scalable CDNs are discussed in chapter 7. This chapter describes our methodology for creating realistic Internet topologies for a set of client sites, the new CDN design model and the optimal solutions for a number of example systems. It also introduces and evaluates the approximate algorithms, which are based on insights drawn from the analysis of simple canonical topologies. Chapter 8 summarizes the dissertation and provides possible directions for future work.

Chapter 2

Related Work

There is a vast literature on traditional web CDNs. However, previously proposed strategies for designing web CDNs are not necessarily efficient for CDNs for streaming media, because they do not account for the unique characteristics of these objects, which were discussed in the previous chapter. More recently, in response to the increasing traffic of streaming media in the Internet, there have been several new proposals for strategies that specifically target the distribution of streaming media objects in the Internet.

Sections 2.1 – 2.3 summarize the main previous contributions for server content, server placement, server selection and routing for unicast streaming CDNs, scalable multicast streaming CDNs and traditional web CDNs. Section 2.4 reviews the previous web and streaming media workload characterizations. As background for a better understanding of the CDN design models developed in this thesis, section 2.5 summarizes previously proposed scalable multicast streaming delivery protocols.

2.1 Server Content

A large number of previously proposed caching strategies for both web and streaming CDNs are designed to react as each new client request arrives, i.e., the policy evaluates whether to cache new

content at the proxy servers every time a new request arrives. We call these caching policies *request driven*. More recently, there have been a few proposals for *rate driven* caching strategies. These strategies determine the server content for a given system configuration, defined by the available resources and current client request *rates* for each object, with the goal of minimizing a given cost function, which is usually a combination of network and server bandwidth costs and, possibly, storage costs. Rate driven policies are a relatively new idea, further explored in this thesis. Relevant previously proposed request driven and rate driven caching strategies are summarized in sections 2.1.1 and 2.1.2, respectively.

Apart from the vast literature on caching strategies, *cooperative caching,* another technique to reduce bandwidth consumption and client latency, by allowing neighboring proxies to cooperate and serve each other misses (i.e., requests for content that is not stored locally) has also been the attention of several researchers. Section 2.1.3 summarizes the previous main contributions in this area. Finally, section 2.1.4 summarizes a few other considerations related to caching of streaming objects, such as smoothing of Variable Bit Rate (VBR) streams and caching from tertiary memory.

2.1.1 Request Driven Caching Strategies

Request driven caching strategies define the content that should be stored at the (proxy) servers dynamically, as requests for new content arrive. In particular, at every arrival of a request to an uncached object, the policy decides whether to cache the object, possibly removing content from the cache to free space. Some of these policies have the additional characteristic of caching the object even if no information about prior client access patterns to that particular object is available. *After* the object is cached, the policies start collecting information about its access frequency rate or recency of access. This information is later used to decide whether that object should be evicted

from the cache to free resources for a new object. Many of the existing caching strategies for streaming and especially for web objects fall into this category.

In the context of traditional web caching, most of the caching strategies are adaptations of traditional memory caching algorithms [130] such as *least-recently-used* (LRU) and *least-frequently-used* (LFU). Because early characterizations of web access patterns suggested the presence of strong temporal locality of reference and preference for small objects [18, 48], most policies combine recency and object size into utility measures that are designed to capture the benefit of retaining a document in the cache [1, 35, 98, 149, 153]. Some of these measures also include other factors such as server latency and bandwidth to the server [35, 98, 153]. In common, most of these policies selectively evict large files from the cache. As an example, the utility measure used by Greedy Dual Size (GDS) caching policy, which delivers very competitive performance [35], is computed as the ratio *cost/size,* where *cost* is a measure of the cost of retrieving the object from the server (e.g., the latency). This measure is decreased over time to dynamically *age* objects in the cache. More recently, new studies [22] reveal a weakening of the strong temporal locality previously associated to Web access patterns. This observation, combined with the Zipfian characteristic of Web access [12, 18, 22, 48, 52], which implies that the probability of future references is dependent on past access frequencies, suggests that the relevance of taking into account long-term access frequencies in caching algorithms is increasing. This fact led to the development of GDSF [16] and GDS-Popularity [86], more efficient strategies that incorporate access counts into the GDS utility measure.

Traditional web caching policies do not take into account the high disk bandwidth requirements of streaming media files. In particular, they typically write new content into the cache every time there is a cache miss. Thus, the cache content is changed very frequently which implies

a high degree of disk write overhead if the policies are applied to streaming media content. Furthermore, these algorithms do not address the special tradeoffs of multicast delivery. In particular, their goal is to maximize cache byte hit ratio. However, maximizing byte hit ratio alone is not sufficient for minimizing delivery cost if scalable multicast delivery is used. Thus, these algorithms are not necessarly adequate for streaming media.

Early work on caching for streaming media propose storing small *intervals* in main memory, so as to satisfy multiple client requests that arrive close in time [50, 51, 53, 54, 90, 107]. Intervals exploit temporal locality and the sequential access pattern of streaming objects to use the limited cache space to store data that is guaranteed to be needed soon in the future at the possible expense of high cache read and write bandwidth requirements. The strategy is to cache the smallest intervals, as they contain the data that will be accessed soonest in the future.

The Resource Based Caching (RBC) algorithm [137, 138, 139] is a complex disk-based policy that caches a mixture of full files and intervals and is guided by two goals: a) keep disk space utilization and disk bandwidth utilization approximately equal and b) replace the entity in the cache that is estimated to be needed again farthest in the future. One key characteristic of RBC is that a new entity is cached only if it can pre-allocate disk bandwidth for it. In particular, the allocation of bandwidth to multiple entities is independent. In other words, bandwidth that is allocated to a cached file cannot be used to serve a request to a different file, even if it is not currently in use. In [138], the authors conclude RBC outperforms the much simpler LFU. However, the design space explored is very limited and some of the results are inconsistent. Therefore, a more thorough analysis of this algorithm is necessary for reaching conclusive results on its performance. Extending the algorithm to incorporate cost tradeoffs for streaming media files that are delivered using scalable protocols is also an open problem.

In [121, 122, 123], Rejaie et al define a frequency-based partial caching architecture and policy for layered encoded video streams [145]. The caching replacement policy is defined together with a mechanism for quality adaptation and congestion control that adds and drops layers in response to variations in the available bandwidth between client and server. A popularity measure, directly proportional to the amount of data delivered per layer, is applied on a per layer basis. Whenever cache space is needed to store a new layer, the policy removes the least popular layer, starting from the last segment. Note that this caching strategy is not adequate to capture client interests for different portions of the files, since it may evict portions of the file that are the most frequently requested if these portions are a small fraction of the file or include the last segments. Other drawbacks of this policy is that it considers only disk space as limited resource (i.e., it assumes unlimited disk bandwidth) and arbitrarily uses disk bandwidth to write any new content (layer or segment) to the cache, which may result in unnecessary write traffic if the content is removed from the cache soon in the future.

2.1.2 Rate Driven Caching Strategies

Rate driven caching strategies determine the server content for a given system configuration defined by the available resources and current client request rates for each object. The content is periodically updated after significant changes in the system configuration and request rates are detected. One example of a simple rate driven strategy for unicast delivery is a frequency-based caching policy that only stores an object in the cache if it is guaranteed to be one of the currently most frequently accessed. This policy, evaluated in chapter 5, maximizes byte hit ratio and thus minimizes the delivery cost. Note that the LFU caching policy, traditionally a request driven strategy for memory caches that stores new content on every cache miss evicting from the cache, if space is needed, the least frequently used objects, may be implemented as this rate driven caching

strategy. Most previous studies that use LFU for web or streaming [3] caching are not clear which implementation is used.

A few early papers propose rate driven caching strategies for unicast streaming [38, 120] and multicast streaming [57, 58]. In [120], multiple caching strategies for unicast streaming are analyzed together with disk striping and replication in a proxy system with a disk array. Using analytical models for disk and streaming performance, the authors compare LRU against the two implementations of LFU discussed above and find that avoiding the arbitrary caching of unpopular files results in performance improvement. This result is consistent with our hypothesis, described in chapter 1. Also in the unicast context, [38] proposes an optimization model for defining the prefix caching strategy that minimizes a cost function including storage and backbone network costs. More recently, optimization models for caching of layered objects with the goal of maximizing a given total revenue function, have also been proposed [91]. As expected for unicast delivery, the best strategy is to cache the content that incurs the highest cost, i.e., the most popular files for minimizing bandwidth cost or the layers with highest revenue per minute of video for maximizing revenue.

As discussed in chapter 1, the shared delivery of a popular object, using multicast, poses several new tradeoffs. In a first attempt to address these tradeoffs, Eager *et al* propose extensions to the scalable dynamic skyscraper protocol [59] to enable proxy caching of file prefixes and formal optimization models that can be solved to determine the full file or prefix caching strategy in the context of a system with homogeneous [57] and heterogeneous [58] proxy servers. The goal is to minimize the total delivery cost, which includes origin and local bandwidth costs, assuming limited cache space and disk bandwidth and that local and origin servers use dynamic skyscraper to deliver the streams to the clients. The authors conclude a caching strategy that allows for caching of file

prefixes is the best strategy compared to retrieving the whole file from one of the servers. In particular, they found that the optimal cache content includes prefixes in the order of 15% of the full media stream for files with a range of access rates in many scenarios. However, the delivery protocol used allows only a very strict partitioning of the data where either a fixed length prefix or the full file might be cached at the local servers. Furthermore, the authors use an approximate estimate of network delivery cost, defined by a single constant to represent the ratio of the average cost of proxy stream to the average cost of an origin stream. Open questions left by this work include how to extend more efficient protocols (such as Bandwidth Skimming [60, 61]) to enable prefix caching, the development of CDN design cost models for these protocols, how to include more precise network delivery costs into these models and whether simple approximate rules of thumb that result in (near) optimal delivery cost can be devised.

The basic formulation proposed in [57] was extended in a series of more recent studies (parallel to this thesis) to develop optimization models for determining the optimal proxy server content for different multicast streaming protocols [76, 118, 143, 144, 147]. The cost models differ with respect to the scalable protocol used, to the assumptions made on the available system resources and to whether the origin uses multicast or unicast delivery. Nevertheless, as in [57, 58], all these works also find (or propose) that a storage strategy that allows for prefix caching is the optimal proxy storage strategy. However, they have some drawbacks. [76, 144, 147] assume limited disk storage but unlimited disk bandwidth, which is not realistic. Furthermore, [118, 143] propose the use of prefix caching and batching of client requests for suffix streams from the origin but do not compare the performance of this strategy against streaming the whole file from only one server or against a delivery protocol that does not rely on batching. Thus, whether prefix caching and batching are the best strategies under flexible and more realistic assumptions remains to be

thoroughly analyzed. In particular, how these two strategies behave if more efficient scalable delivery protocols (such as Bandwidth Skimming [61]) are used remains an open question.

2.1.3 Cooperative Caching

Cooperative caching is a technique where neighboring proxies serve each other misses (i.e., share their contents) in order to reduce the load on the network and on the origin server. Cooperative web caching was first proposed in the context of the Harvest project, with the design of the Internet Cache Protocol (ICP) [84] that supports discovery and retrieval of documents from neighboring caches. The more scalable and efficient *Summary Cache* [66] was later proposed. Cooperative caching is not addressed in this thesis as our focus is on hierarchical CDNs. However, the caching policies used in the cooperative caching architectures reviewed in this section are more closely related to this thesis.

Several architectures for cooperative caching of streaming media have been proposed [3, 24, 37, 74, 80, 95]. The common strategy is to partition the file into several segments and spread them across multiple servers. In [3], the authors use a set of request traces [4] to compare several caching policies for their *MiddleMan* cooperative caching system. They found that LRU-k [106] and LFU outperform LRU and FIFO. The LRU-k policy is a request driven replacement policy that evicts from the cache the object that has the largest k^{th} distance, defined as the time since the k^{th} most recent client request. The performance comparisons are performed under the assumption of unlimited disk bandwidth. In particular, all policies cache a file on a miss, which might hurt performance under more realistic assumptions if disk bandwidth is wasted on caching unpopular files. LRU-k is also used in [95] as the replacement policy in a system that combines prefix and segment caching with video summarization, where key representative frames are stored at each

neighboring proxy so that clients may quickly choose which portion of the video he/she wants to retrieve.

The caching policy used in the SOCCER system [24, 80] is not well defined but includes interval caching, segment caching and prefix caching (inside a segment) and the system works with either unicast or multicast delivery. Unlike SOCCER and Middleman, RCACHE [37, 74] caches multiple replicas of the same segment in different servers. The number of replicas is defined by the segment popularity. In particular, a segment may not be stored in any server if it is not one of the most popular segments. Cooperative caching has also been proposed together with active routers, placed across the network, which actively store data that is captured while being transmitted from the origin server to the clients and cached using either interval [141] or segment caching [136].

2.1.4 Other Streaming Media Caching Considerations

In addition to reducing bandwidth costs and client latency, partial file caching can also be applied for reducing variability in network resource requirements. Previously proposed techniques include prefix caching [128] for smoothing playback of a variable bit rate streams, video staging [148], which caches the bursty portions of popular files, and *selective caching* [105], which caches prefix and intermediate segments for limiting the probability of client buffer underflow. This work is orthogonal to this thesis, which develops caching strategies for minimizing delivery cost. However, our solutions can be extended to include one of these techniques.

Given the large sizes of video files, there have been proposals for a hierarchical storage system consisting of both tertiary and secondary memory such as the Berkeley VOD System [28], where video files are permanently stored in tertiary memory (tapes) and are brought to hard disk in

response to client requests. Analytical models for designing the architectural parameters of such systems (tertiary, secondary, network bandwidth and disk storage) have also been proposed [39].

A different approach is taken in [117], where the authors propose a memory caching strategy that stores data in a pre-packetized format ready for immediate transmission with the goal of avoiding the overhead associated to user level processing and assembly of packets.

One final issue that is related to the design of caching strategies is the design of block allocation and placement strategies for improving disk throughput and load balancing across multiple disks. Among the previously proposed strategies are disk striping [108] and random data allocation [127].

2.2 Server Placement

Another key challenge for designing efficient CDNs is to define where replicated content should be placed, or in other words, to define the number and placement of (proxy and origin) servers in the network. More specifically, the server placement problem consists of determining, for given network topology and, possibly, set of client populations, where a number of servers should be placed in order to minimize an arbitrary cost function (e.g., bandwidth costs, client latency). This problem can be modeled after a well-known problem in graph theory, the center placement problem, or any of its variations, the minimum *k*-median problem or the facility location problem. An optimal solution for this problem has been shown to be NP Complete for general graphs [70]. Hence, previous works in this area either address this problem for specific topologies, such as daisy chains, rings [93] or trees [93, 96] or propose approximate algorithms for the general case [85, 115, 116]. An extensive survey of previously proposed algorithms for server placement is given in [92]. This section summarizes some of the most relevant previous contributions.

The problem of placing web proxies with the goal of minimizing client latency for a target server was first analyzed in [96]. This problem is mapped into the placement of proxies in a tree rooted by the server. The authors provide an optimal dynamic programming solution with polynomial complexity. A similar problem was later investigated by the authors in [93], who provide a more efficient optimal solution. They also propose a greedy algorithm that places replicas one at a time, choosing at each step to place the new server at the node that minimizes a formulation of the total cost. They show that their greedy algorithm performs very close to the optimal in many Internet-based tree topologies.

More recently, several researchers addressed the placement problem for more general, random or Internet, network topologies [85, 115, 116]. They propose and compare a set of approximate algorithms, including the previously proposed greedy algorithm [85, 115], a fan-out based placement strategy which places servers at the nodes with highest degree of connectivity [85], hot-spot placement, which places servers close to the clients with higher demand [115] and random placement [85, 115]. The key conclusions are that a) the greedy algorithm performs very well for general topologies [85, 115] and b) the fan-out based placement algorithm performs very close to the greedy algorithm with the advantages of lower complexity and no requirement of prior knowledge about the clients [85]. More recently, Radoslavov *et al* analyzed the sensitivity of the fan-out based placement algorithm to the level of abstraction of the nodes selected for placement in Internet topologies [116]. The key result is that not only the servers should be placed at the most connected Autonomous Systems (higher level of abstraction), as proposed in [85], but also they should be placed at the most connected routers inside the selected Autonomous Systems.

All these previous and parallel studies apply the proposed algorithms for unicast delivery, and assume fixed server selection and routing strategies, namely each client contacts the closest replica

server and the responses take the shortest path back to the clients, which are the optimal strategies for unicast delivery. Whether these conclusions hold for multicast delivery is not clear since none of the policies directly accounts for savings from sharing of multicast streams. Server placement for CDNs that use scalable multicast delivery have been explored only in a very restricted form where the placement is fixed and the number of replicas is either equal to one (the origin server) or to a fixed number (the proxy servers) [57, 58, 118, 144, 147].

We point out that most of the studies, summarized in this and in the following section, use either randomly generated or Internet network topologies. Despite the large number of topology generators [6, 8, 29, 34, 89, 102, 154], a number of studies focus on analyzing characteristics of the Internet (e.g., connectivity) and on generating Internet topologies from publicly available or collected information [26, 41, 65, 68, 72, 133].

2.3 Server Selection and Routing

For unicast delivery, contacting the "closest" server and routing through the "shortest" path are the optimal strategies, assuming path distances that accurately reflect network cost. The focus of most previous works is on defining how to estimate path distance, using either static or dynamically collected measures. As an example, static shortest path or weighted shortest path, measured by the number of (weighted) hops, are the primary routing algorithm in use in today's Internet for both unicast and multicast delivery [49, 63]. The use of static measures of geographical distance for routing has also been proposed in [77]. Other routing strategies make decisions based on dynamically collected measures of the current state of the network and servers [13, 14, 36, 71, 75, 110]. Furthermore, locality aware strategies, which target both load balancing and throughput, explore temporal locality by sending to the same server, requests to the same object that

arrive close in time as long as the load imbalance among the servers is kept under a certain threshold [71, 110].

In [30], server selection is studied in the context of downloading of large files in peer to peer systems operating in an overlay network. The goal of this work is to increase throughput by letting clients simultaneously download different portions of a file from different peers. This work focuses on designing efficient algorithms for peer selection and for reconstructing the original file.

In the context of streaming, routing strategies have been proposed with the goal of improving content quality and resilience to packet losses by the use of redundant delivery paths [15, 73]. This is orthogonal to the goal of this thesis (i.e., to minimize delivery cost). However, the methods developed in this thesis can be extended in the future to include strategies that also target improving resilience to packet loss.

To the best of our knowledge, server selection and routing strategies that specifically account for savings from multicast have only been partially addressed in two recent, parallel, studies [67, 155]. In [67], the authors propose optimization models and heuristics for server selection assuming fixed server location and a single multicast stream, that has network bandwidth independent of the number of clients receiving the stream, which is the case for broadcast or live streams. In [155], on the other hand, the authors propose two routing heuristics for the scalable bandwidth skimming multicast protocol, which has server bandwidth requirement that is dependent on client load. However, the heuristics are evaluated on randomly generated network topologies with only one server, with fixed placement.

Server selection and routing for CDNs that use multicast delivery is further explored in this thesis. In comparison with [67], we address the problem for on-demand delivery of stored content using the efficient scalable bandwidth skimming protocol, which has bandwidth requirements

dependent on client load. Our work builds on the results from [155] and derives a set of approximate algorithms for both placement and routing. Some of the routing algorithms are similar to those developed in [155] but are generalized for the case of multiple servers. We also provide an optimal algorithm and use it as benchmark to evaluate the approximate algorithms in a number of realistic Internet topologies.

More generally, a common strategy among all previous works is to address the server selection and/or routing problem assuming fixed server location. Similarly, as described in the previous section, server placement has been addressed assuming shortest path routing, which is the optimal strategy for unicast delivery. This is a key point that distinguishes our work from the previous ones. We solve the joint server placement and routing problem for scalable CDNs, where the optimal strategy for either problem is not known a priori. This significantly increases the complexity of our solutions, compared to previously proposed methods.

2.4 Workload Characterization

A clear understanding of characteristics of the workload such as client request arrival process and object popularity is very relevant to the design of efficient CDNs. The statistical properties of web workloads have been investigated in a number of studies. Mostly based on log analysis, they include characterization of client [22, 48], proxy [25, 35] and web server [17, 18, 20] workloads. The logs used in these studies are composed most entirely of web files. Given the distinct and unique characteristics of streaming media objects, it is not clear whether their conclusions hold for a workload containing a larger fraction of streaming content. More recently, in response to the growing popularity of streaming content in the Internet [2, 152], there have been a few attempts to characterize streaming media workloads [4, 42, 44, 78, 79, 104, 109, 142], many of which are parallel to this thesis. The workloads analyzed in these studies differ in several dimensions: server

versus client workloads, stored versus live content, entertainment, educational or commercial content. This section summarizes the main previous findings regarding several streaming workload characteristics. Whenever appropriate, the corresponding findings for web workloads are also reported. These observations are used in the development of workload generators such as Surge [20] (for web workload) and GISMO [87, 142] (for streaming workload).

Despite a large variation in object sizes and encoded bit rates, one common finding among all previous characterizations of streaming workloads is the large fraction of client sessions that retrieve only a small portion of the file, reflecting very interactive client populations [4, 42, 44, 78, 79, 104, 109]. Furthermore, the distribution of session lengths for different workloads has been shown to have a heavy tail [44, 103] and has been characterized as either Gamma [109] or Lognormal distributions [109, 142]. Heavy tailed distributions have a high variability and, in practical terms, indicate that very large values are possible with non-negligible probability. Client sessions are further broken into a sequence of client interactions including pauses, resumes, fast-forwards, rewinds or jumps into specific points of the video. Such client behavior is modeled as a sequence of media alternating ON and OFF times and models for the distribution of both ON and OFF times are provided in [109, 142]. In the context of web workloads, there is no concept of partial access. However, the distribution of file sizes, including files stored on servers, requested by clients and transmitted over the network, has also been characterized as heavy-tailed, usually modeled with a Pareto distribution [17, 18, 20, 22, 47, 48].

The distribution of inter-arrival times for streaming workloads has only been analyzed in a few studies [4, 103, 142]. The attempt to characterize inter-arrival times in [4] resulted in no clear indication of any particular distribution, except that it seems to depend on the category of the file (entertainment versus educational content). One drawback of this analysis is that it was performed

over data collected during a period of six months. Therefore, the effects of data aggregation over such an extended period during which the arrival process is not likely to be stationary may have hidden the true distribution. The authors in [103] are more successful in showing that although the request inter-arrival times seem to be heavy tailed, the session inter-arrival times follow an exponential distribution. Similar results have also been observed for web workloads: whereas the inter-session times are usually modeled with an exponential distribution [18, 112], the inter-request times are usually modeled with a heavy tailed Pareto distribution [18, 20, 47]. In contrast, the distribution of inter-arrival times for both sessions and requests for a live streaming workload has been found to be exponential over periods of stationary load in [142].

File popularity is another characteristic that has been analyzed in many previous works. In the web context, it has been consistently characterized by a Zipf or a Zipf-like distribution [156], which relates frequency of access to popularity rank via a power-law relationship, i.e., Prob(access file i) = C/i^{θ}, where $\theta > 0$ and C is a constant to guarantee the sum of the access probabilities over all files is equal to 1 [12, 17, 18 ,20, 22, 25, 48]. In the streaming context, a few previous studies have successfully characterized file popularity with a Zipf-like distribution [42, 44, 52]. In [42], the authors measure file popularity over different time scales (a month, six months, one year) and conclude that a Zipf-like distribution models file popularity of their workloads reasonably well if the data is analyzed over the right aggregation period. A Zipf-like distribution has also been used, in the context of live streaming, to characterize client interest profile, defined as the number of times a client access the same live stream related to the same live content [142].

Another aspect of a workload that may have implications for caching policies is temporal locality. Temporal locality refers to a characteristic in which there is a higher probability of referencing in the near future objects that have been referenced in the recent past and has been the

focus of attention of several previous studies, especially for web workloads [4, 12, 17, 20, 22, 25, 48, 86, 100]. They all observe strong temporal locality in the workloads analyzed, although [22] shows a recent weakening in the temporal locality of a web workload.

Finally, three previous works provide analysis of streaming traffic from a network standpoint focusing on characteristics such as packet inter-departure time [103], spatial workload distribution [104], packet loss, delay jitter, packet reordering and path asymmetry [97].

Although collectively previous and parallel analyses of streaming workloads cover a large set of different characteristics, most of them focus only on a few specific aspects of the workload. Thus, there is a need for a more thorough characterization covering most of the relevant aspects for the same streaming workload. There is also a need for a careful analysis of how each workload aspect considered varies over time. In particular, insights into how client request rate and file access frequency change over time are important to determine how frequently a CDN configuration should be updated. Futhermore, analyzing the time varying aspect of the characteristics is also important in order to identify periods of approximate stationary behavior, over which the characterization of the workload aspect should be carried out. It is important to avoid analyzing data over periods of non-stationary behavior. This is one key point that distinguishes our work from previous ones. It is also explored in two parallel works [42, 142].

2.5 Scalable Streaming Media Delivery Protocols

Recently, a number of scalable multicast streaming protocols for delivering popular stored media content have been proposed with the promise of significant reductions in server and network bandwidth, compared to the conventional unicast streaming. This section summarizes the key

characteristics of some of these protocols and describes Bandwidth Skimming, a more efficient scalable protocol that is the basis for many of the techniques developed in this thesis.

The simplest multicast streaming protocol, *batching*, consists of making clients wait in the hope that enough client requests are received within a certain period. These requests are batched together and served using a single multicast server stream [55]. Although simple, this protocol may incur a very long client waiting time. Another technique is piggybacking [5], where the client speeds up or slows down the rate at which the file is displayed in order to bring different streams to the same file position, at which time the streams merge, reducing bandwidth requirements. However, the efficiency of the protocol is limited by the maximum change in the display rate that a client can tolerate (± 5%). For both batching and piggyback, the required client bandwidth is limited to the file streaming rate.

If clients have bandwidth larger than the file streaming rate and enough buffer space, more efficient protocols, which let a client simultaneously receive data from multiple streams, can be used. The client shows the data received in one of the streams to the user as it is received and simultaneously buffers the data received in the other streams, which allows it to catch up and share the transmission of the rest of the file with an earlier client.

One class of such protocols is periodic broadcast [5, 57, 58, 59, 83, 111]. In this protocol, the server divides the file into segments and continuously broadcasts them on a fixed number of transmission channels (or streams). Upon arrival of a client request, the server returns a schedule for the client to tune into each of the transmission channels to receive each segment in time for playback to the user. An example of periodic broadcast is the static skyscraper algorithm [83], which divides the file into increasing segments and requires that clients must be able to receive on two channels simultaneously. Because the segments have increasing size, clients that receive

different early segments will batch together for receiving later longer segments. Since server bandwidth requirement for delivering a given file is constant, this technique is very efficient for large request rates. The dynamic skyscraper [59] improves on this technique for the case of time varying file popularity and for providing true on-demand service. The partitioned dynamic skyscraper further improves the protocol to include proxy caching of earlier segments [57, 58]. The digital fountain approach [31] is a different broadcast protocol designed for reliable download of bulk data over lossy channels.

Another class of delivery protocol is patching [33, 82, 129]. Upon arrival of a client request, patching immediately starts multicasting the object. A later arriving client joins the on-going multicast stream and receives the initial (missed) portion of the file from a separate *unicast* stream. The algorithm limits the bandwidth required for unicast streams by using a threshold parameter. If the missed portion of the file is longer than the threshold, the server creates a new multicast stream to deliver the whole file. Catching and selective caching [69] are variants of this protocol that combine both patching and periodic broadcast.

Bandwidth Skimming [60, 61, 62] is a more efficient scalable streaming protocol, based on hierarchical merging of multicast streams to reduce bandwidth. The following sections describe the basic Bandwidth Skimming protocol in details, as it is the basis for many of the techniques developed in this thesis.

2.5.1 The Bandwidth Skimming Protocol

The basic idea behind Bandwidth Skimming is that upon arrival of a new client request, a new *multicast* stream is created to deliver the file. In one variation of the protocol, called Closest Target, the client listens to the new multicast stream as well as to the closest (target) multicast stream that

is still active. When the new stream has delivered all of the data that the client missed in the target stream, it is terminated, and all clients listening to the target stream are now "merged". The clients listening to the "merged" stream also start listening to its closest previously created multicast stream that is still active (which becomes its target). However, if the target stream terminates before the later stream is ready to catch up to it, the clients listening to the later stream simply start listening the next closest target stream. Similar protocols based on hierarchical merging are also introduced in [21, 46].

Unlike previous scalable streaming protocols, bandwidth skimming can also be used if client bandwidth is less than twice the streaming rate. Clients use their available bandwidth to listen to an increasing *fraction* of the target stream (and a corresponding decreasing fraction of their main stream) in order to merge with the target stream [61]. Thus, bandwidth skimming can be used to deliver *high quality* videos that require a significant fraction of the client bandwidth. Furthermore, bandwidth skimming works well in the presence of client interactivity, is more scalable than patching, is efficient for a wider range of request rates than periodic broadcast. Moreover, Bandwidth Skimming has been modified for reliable delivery of content over lossy channels by including redundant data that is computed using erasure codes [101].

2.5.2 Required Server Bandwidth for Bandwidth Skimming

Because streams are merged hierarchically, the average server bandwidth required to provide immediate service to each request grows only logarithmically, with a small constant factor, in the client request rate. The following formula has been shown to be a good estimate of the average required server bandwidth measured in units of the streaming rate if client bandwidth is equal to twice the streaming rate, arrival process is Poisson and each client requests the entire file, stored at the server [62]:

$$B(N) = \eta \ln (1 + N/\eta) , \quad (1)$$

where N is the average number of simultaneous requests received during the playback of the file and $\eta=1.62$ if client has enough bandwidth for receiving two simultaneous streams. Values of η for client receive bandwidth lower than two streams are derived in [62].

Bandwidth Skimming is more efficient and scalable than previously proposed delivery techniques. The required average server bandwidth is very close to the minimum required for immediate service [61] over a wide range of request rates. In comparison, the required average server bandwidth for the Patching protocol increases with the square root of the request rate [62].

Chapter 3

System Design Considerations

A streaming CDN architect needs to take into account the following key issues to determine the most cost-effective solution for the four CDN design problems (i.e., server content, server placement, server selection and routing) defined in chapter 1: a) the architecture of the distribution network (e.g., hierarchical or peer to peer), b) the streaming protocol used by each (origin and proxy) server (e.g., unicast, periodic broadcast, bandwidth skimming), c) characteristics of the client workloads such as the client request rates for different objects and how frequently these rates change and d) the storage capacity, disk bandwidth and network i/o bandwidth available at each server as well as the network bandwidth available on the path(s) between server(s) and clients.

As mentioned in chapter 1, the focus of this thesis is on developing methods for hierarchical CDNs. This chapter elaborates on the other issues. Section 3.1 further describes the components of a CDN. The content delivery cost functions that are minimized in order to determine the optimal solutions for the scalable CDN design problems addressed in this thesis are provided in section 3.2. The set of system and workload parameters and assumptions used in developing scalable CDN design models and approximate algorithms and the caching algorithms for conventional unicast CDNs are discussed in section 3.3. Section 3.4 discusses our approach to explore a large region of the CDN design space defined by these parameters and assumptions.

3.1 Further Description of CDN Components

The operation of a CDN was briefly discussed in chapter 1 (section 1.1). Recall that, a CDN consists of a group of clients, a group of (proxy and origin) servers where content is replicated and a network connecting servers and clients. This section elaborates on each of these components and on how they are defined in the context of the solutions developed in this thesis.

For instance, our algorithms and CDN design models are solved for a set of client sites, instead of considering each individual client separately, where a client site is defined as a group of clients that are located geographically close to each other and are connected to the same given access point (e.g., a cable network, a domain in the Internet). In practical terms, this means that our methods consider the statistically aggregated content requests from a group of clients, instead of each individual client request.

We consider CDNs where origin and proxy servers use either unicast or a scalable multicast delivery protocol. For scalable delivery, the Hierarchical Streaming Merging Bandwidth Skimming [60, 61] is the streaming protocol assumed, unless otherwise stated. Bandwidth Skimming was chosen because of its scalability properties, superior to other protocols such as patching [33, 82, 129], and its overall high efficiency for all ranges of request rates. Furthermore, it can be used in interactive environments and has been extended to include techniques for packet loss recovery [101]. Although our strategies assume the basic bandwidth skimming protocol, the approach, main insights and conclusions obtained should also be valid if the reliable bandwidth skimming is used. The underlying multicast protocol used by each server may be either IP multicast [63], which may become more widely available as new protocols supporting single-source multicast are implemented [81], or application level multicast [45].

As discussed in chapter 1, client requests are sent to a proxy server, typically the closest one, which may forward them to the origin server, in case the requested content is not (fully) stored locally. We point out that the origin server may stream the content back either to the proxy server, which then forwards it to their clients, or directly to the clients. Appropriate scenarios will be considered in evaluating caching algorithms and designing the minimum delivery optimization cost models. In particular, we point out that it is cost-effective to transmit origin streams through a proxy only if the content is simultaneously being written into the proxy cache or if the origin server uses unicast and the proxy uses multicast, in which case, it results in significant bandwidth cost reductions. Otherwise, it is more cost-effective to stream the origin content directly to the client as it results in lower server and, possibly, network bandwidth costs.

The network topology connecting clients and servers (e.g., the Internet) can be modeled at different levels of abstraction. At a higher level, the backbone or remote network and each proxy (local) network are represented as single nodes. For example, the network in Figure 1 (section 1.1) contains three nodes. More detailed hop-level topologies with server access points can be specified depending on how network bandwidth cost is charged. As discussed in chapter 1, appropriate charging schemes for multicast streams on the Internet are still an open problem. A methodology for creating hop level Internet topologies is proposed in chapter 7.

3.2 Scalable CDN Design Cost Models

For scalable CDNs, this thesis develops analytical models for deriving the minimum delivery cost design. In other words, we develop analytical models to compute the proxy server content, (proxy and origin) server placement, selection and routing that minimize the total delivery cost. This section defines two formulations of the delivery cost for a CDN as well as briefly introduces the CDN design models proposed. A detailed description of these models is given in chapters 6 and 7.

The total delivery cost of a CDN, which corresponds to the cost, ultimately paid by the content provider, to have its content delivered to the clients, includes server and network delivery costs. Server and network delivery costs are *proportional* to the bandwidth that server and network must be provisioned with in order to handle the expected workload, respectively. The cost of other server resources (i.e., processors, disks and faster network interface cards) is included as part of the server bandwidth cost since additional resources are needed as the required server bandwidth increases. An important previous result that relates the required server (and network) bandwidth that a CDN must be provisioned with to handle a given load with the required *average* server (and network) bandwidth was obtained in [134]. For a given client load, Tan *et al* show that the amount of bandwidth that a server that delivers a number of files should be provisioned with to guarantee a very low client balking probability (i.e., one in ten thousand), i.e., to guarantee that a client interrupts its request before being served with very low probability, is approximately the same as the *average* concurrent server bandwidth required to sustain that load [134]. This is because the variation in the sum, over all files, of the server bandwidths needed to give every client immediate service is statistically lower than the variation in the server bandwidths for each file. Intuitively, during periods when a file requires more than its average server bandwidth, other files may use less than their average server bandwidth. This result implies that assuming a server (and network) will be well provisioned for the anticipated client load, server (and network) delivery cost is proportional to the average server (and network) bandwidth required to sustain that load. This is a key observation, on which our models and solutions are based. We will use formulations of the delivery cost that are proportional to the average server and network bandwidth.

The total average (or provisioned) network bandwidth includes average remote network bandwidth to transmit the stream from the origin to the region where the client is located and

average local network bandwidth to transmit the stream over the "local" network where the client is located. Note that the cost of a proxy stream includes server bandwidth costs and local network bandwidth costs whereas the cost of an origin stream includes server bandwidth costs, remote network bandwidth costs and, if the origin streams the content directly to the clients, local network bandwidth costs.

This thesis uses two formulations for the total delivery cost, one of which was previously proposed [57, 58] and the other is new. The previously proposed formulation assumes network delivery cost can be computed directly from the average number of (multicast) streams that use the network concurrently. More specifically, the remote network bandwidth, expressed in units of the average streaming rate, is the average number of origin server multicast streams and the local network bandwidth, in units of the average streaming rate, is the average number of proxy multicast streams. In that case, the network topology can be represented at the highest level of abstraction where "remote" and "local" networks are represented as single nodes, as the optimal solution depends only on the ratio of the average cost of a proxy server stream to the average cost of an origin server stream, rather than the exact network paths used by each stream. The delivery cost, D_1, defined above, is expressed, for P proxy servers, as:

$$D_1 = B_{origin} + \sum_{p \in P} \beta_p \times B_{proxy\_p}\,, \tag{2}$$

where B_{origin} is the total average concurrent server bandwidth required to deliver the content from the origin server, B_{proxy_p} is the total average concurrent server bandwidth required to deliver the content stored at proxy p and β_p is the ratio of the average cost of one proxy p stream to the average cost of one origin stream. The formulas for B_{origin} and B_{proxy_p} depend on the streaming

protocol used by the origin and by proxy *p*, the client load at each proxy and on the content replicated at each proxy server.

To design the CDN, equation (2) is minimized over all possible number and placement of proxy servers, delivery trees and (partial) media content that could be stored at each proxy, subject to constraints on the storage and bandwidth available at each proxy. This model was first proposed in [57, 58] where the authors developed specific server bandwidth formulas and results, assuming the dynamic skyscraper protocol. In chapter 6, we will build on this model and develop protocols, bandwidth formulas and results for the more efficient bandwidth skimming protocol, which, unlike the dynamic skyscraper, allows an arbitrary file prefix to be stored at each proxy.

The multicast streams transmitted to different sets of clients may be transmitted over different network paths, incurring different network costs. We propose and develop a new more elaborated formulation of the total delivery cost that takes into account the network path traversed from server(s) to client sites to include a more detailed calculation of network bandwidth costs. In particular, the total average network bandwidth of a CDN is defined as the (possibly weighted) sum of the average concurrent network bandwidth on each hop in the network used by each server to deliver content to the client sites. This formulation of the delivery cost, D_2, is expressed as:

$$D_2 = \sum_{(x,y)} w_{x,y} \times B_{networkhop\_x,y} + \gamma \sum_{j} B_{server,j} \ , \qquad (3)$$

where $B_{networkhop\_x,y}$ is the average bandwidth required for the network hop between nodes x and y, $B_{server,,j}$ is the average bandwidth required for (origin or proxy) server j, γ is defined as the ratio of the cost of one unit of server bandwidth to the cost of one unit of per-hop network bandwidth and $w_{x,y}$ is a weight assigned to the network hop between x and y to represent the relative cost of one unit of bandwidth among different network hops. To design a CDN, an optimization design model

can be constructed where equation (3) is minimized over all possible number and placement of servers, server content and delivery trees with possible constraints on server storage and bandwidth as well as network bandwidth on each hop. This model is defined more precisely and further explored in chapter 7.

The optimization cost models for designing scalable CDNs developed in this thesis, are implemented using the General Algebraic Modeling System (GAMS) system [27]. GAMS is a modeling system for mathematical programming problems which includes a language compiler and a set of efficient solvers. The cost models developed in this thesis contain non-linear and non-convex functions. For these functions, the optimization algorithms do not guarantee true optimality. However, we have modified these models, as described in chapters 6 and 7, to include a mixture of linear functions and integer binary variables, for which several solvers can be used to find the global optimal solution. In particular, we use the CPLEX solver, included in the GAMS system, to solve our linear models.

3.3 Workload and System Configuration Assumptions and Parameters

The set of basic system and workload parameters that are used in the optimization CDN design models and algorithms, as needed, to determine the optimal number of proxy servers, server placement, routing and proxy server content is given in Table 1. The table also includes the outputs from the optimization models and algorithms. We claim that the parameters in Table 1 represent all the essential properties of the system, with the exception of partial file accesses due to client browsing or other interactive behavior such as pause, fast forward, skip ahead, and so forth. However, the CDN design models, algorithms and solutions derived for the given parameters that assume only full file accesses can be refined and applied to systems with interactive client loads, as

Table 1: System and Client Workload Input Parameters and Outputs of the CDN Design Methods

System Parameters	**Definition**
P_{space_p}	Storage capacity available at proxy p
P_{bw_p}	Streaming bandwidth available at proxy p
V	Set of nodes {1, 2, 3, … v} that define the network topology
A	Set of *directed* arcs $\{(x,y),\ x, y \in V\}$ in the network topology
$NetBW_{x,y}$	Bandwidth available in the network hop represented by the arc $(x,y) \in A$
$w_{x,y}$	Weight on arc $(x,y) \in A$ (i.e., relative cost of one unit of bandwidth on the arc)
S	Access points for server placement $(S \subseteq V)$
C_V	Client sites $(C_V \subseteq V)$
β_p	Ratio of the average cost of one proxy p stream to the average cost of one origin stream
γ	Ratio of the cost of one unit of server bandwidth to the cost of one unit of network bandwidth over a hop
b	Minimum bandwidth available at the client, in units of the average streaming rate
Workload Parameters	**Definition**
n	Number of media objects or files
T_i	Duration (in minutes) of file i
r_i	Streaming rate (in Mbps) of file i
λ_c	Total client request arrival rate from client site $c \in C_V$
$p_{i,c}$	Access probability for file i from client site $c \in C_V$
$\lambda_{i,c}$	Total client request arrival rate for file i from client $c \in C_V$ $(\lambda_{i,c} = \lambda_c \times p_{i,c})$
Models Outputs	**Definition**
P	Number of proxy servers
S^*	Set of (proxy) server nodes in the optimal delivery network $(S^* \subseteq S)$
$f_{i,p}$	Fraction of file i stored at the proxy server p
B_{proxy_p}	Server bandwidth required to deliver the content stored at proxy p
B_{origin}	Server bandwidth required to deliver content from the origin
$N_{x,y}$	Average concurrent number of streams traversing the hop represented by arc $(x,y) \in A$
$B_{network}$	Total average network bandwidth summed over all hops
B_{server}	Total average concurrent server bandwidth summed over all servers

explained below. The parameters in Table 1 define a list of implicit assumptions on the system operation, which are discussed next.

Unless otherwise stated, we assume media request arrivals, overall and per file, are Poisson, as observed for web workloads [18, 112] and for some streaming media workloads [103, 142]. The request arrival rates from each client site c may change over time and the parameter λ_c represents the instantaneous rate at a point in time. We point out that the distribution of inter-request arrival times for other web workloads [20, 112, 150] have been observed to have a heavier tail than exponential reflecting a more bursty arrival processes. The more bursty arrival process has three principal effects on the design of streaming media CDNs. First, interval caching may be more efficient as the intervals will be smaller, on average. Second, the total average concurrent bandwidth used to deliver a media file using a scalable streaming protocol decreases, because streams merge more quickly, on average. The decrease in bandwidth is relatively small [62] and applies to (origin and proxy) server bandwidth and to (remote and local) network bandwidth, and thus does not impact the optimal CDN design, which depends more strongly on the *relative* cost of a unit of server bandwidth versus a unit of (remote) network bandwidth. Third, given the average concurrent bandwidth to deliver a full file using a scalable streaming protocol, the fraction of this bandwidth needed to deliver a small prefix is larger for Poisson than for more bursty arrival processes. Thus, the optimal proxy content for Poisson arrivals may contain more prefixes than for more bursty arrivals. On the other hand, if the optimal proxy content for Poisson arrivals includes only full files, then the same solution is likely to hold for more bursty arrivals. We examine the request arrival process for two media servers in chapter 4.

Client access frequency is defined per file. In practice, due to client browsing behavior, different segments of a file may have different access frequencies. Interactive file accesses have only been characterized statistically (e.g., percentage of fast forwards, percentage of rewinds, average skip ahead distance) and for a small number of media servers in the literature [78, 79, 109].

There are potentially many different interactive access patterns for a given file that could occur in practice, such as decreasing frequency of access with increasing position in the file (as clients decide whether to continue playing the media), equal frequency of access as a function of the position in the file if clients primarily pause and resume, rather than rewinding and fast forwarding, and higher frequency for one or more particular segments that, for instance, have high entertainment appeal or that are particularly important points in an educational client. This thesis provides a more detailed analysis of file access patterns for one media server in chapter 4.

There is one principal impact of interactivity on the CDN solution, that is, content placement and storage decisions should be applied to file segments, rather than to full files. We point out that short requests and other segments that have low enough access frequency require unicast streaming solutions rather than scalable solutions. Scalable delivery applies to the longer or more popular segments, although it may be somewhat less efficient for the segment requests than for full file requests. Thus, to a first order, models, algorithms, solutions and insights derived for workloads with full file access of varying frequencies can be applied to segments that have approximately uniform access frequency. In addition, they can be refined to include more precise measures of the partial file access behavior, as discussed in chapter 8. For these reasons and because parameters that capture partial file access pattern cannot be defined until client interactive access patterns have been characterized for actual workloads, this thesis develops models and algorithms for workloads with full file accesses.

Like the request arrival rates, file access frequency may also change over time and $p_{i,c}$ represents the access probability for file i from client site c at a point in time. The goals of this thesis are to obtain (a) insights into how frequently the file access frequencies and request rates change for two media server workloads (chapter 4) and (b) derive CDN design models that can be

used to obtain the optimal solution for each period over which the designer determines that the CDN configuration will be static. These intervals of time are such that it is too expensive (and thus not cost-effective) to update proxy server content, placement or routing. To derive results from our proposed CDN design models, we will assume file access frequencies are given by a Zipf-like distribution [156] as observed in a number of previous analyses of web and media workloads [12, 17, 18, 20, 22, 25, 48]. Recall that, in a Zipf-like distribution, Prob(access file i) = C/i^{θ}, where $\theta > 0$ and C is a constant to guarantee the sum of all file access probabilities is equal 1.

The streaming protocols considered assume a *minimum* client bandwidth equal to b. We point out that, although the models and algorithms developed in this thesis assume full file accesses, they can be directly applied to file layers (instead of full files), in case clients have heterogeneous bandwidth and file layered encoding is used.

We assume files are placed on the proxy disks in a way that balances the load across the disks [108, 127]. That is, the total disk bandwidth of each proxy p is assumed to be available for writing into and reading from the proxy disk cache. We note that, in practice, the available disk bandwidth may be lower than the manufacturer reported values due to disk access overheads and delays. However, if efficient disk block allocation and placement methods, such as those proposed in [108, 127] are used, we expect the ratio of the total disk space to the total disk bandwidth, which is the disk parameter that primarily impacts the optimal proxy content, to remain roughly the same.

3.4 Derived Workload and System Configuration Parameters

To evaluate CDN design solutions and proxy caching algorithms, Table 1 provides a daunting list of parameters to be varied, each of which affects CDN performance. Previous and parallel studies have set each of most of the parameters to a particular fixed value, and independently varied a

small number of the parameters, such as total average request rates, proxy cache size and maximum proxy bandwidth. In some cases, these few parameters are also only varied over narrow ranges of values (e.g., total cache size smaller than 16GB [138]) or some of the parameters are set to unrealistic values (e.g., limited proxy disk space but unlimited proxy disk bandwidth [76, 144, 147]).

A key goal of this thesis is to evaluate CDN solutions over as much of the system design space as is practically feasible. To achieve this goal, we first identify, in section 3.4.1, a subset of all parameters that have primary impact on a CDN delivery cost and thus need to be varied to explore the design space. Next, we make the key observation that not all of these parameters have an independent impact on the CDN performance. For example, total proxy storage capacity and the sum of the media file sizes do not need to be varied independently, because the optimal proxy content depends primarily on the ratio of these two parameters. Finally, we define a smaller set of derived and normalized parameters that will be varied to evaluate the quality of the CDN designs in this thesis. The relationships among the primary parameters are discussed in section 3.4.2 and the derived parameters are given in section 3.4.3.

3.4.1 Primary CDN Design Evaluation Parameters

The basic system and workload parameters that could have a primary impact in the relative performance of a CDN solution include:

(1) storage capacity available each proxy, i.e., $\{P_{space\_p}, p = 1, \ldots, P\}$,

(2) streaming bandwidth available at each proxy, i.e., $\{P_{bw\_p}, p = 1, \ldots, P\}$,

(3) number n of files,

(4) file durations, i.e., $\{T_i, i = 1, \ldots, n\}$,

(5) client request rate from each client site, i.e., $\{\lambda_c,\ c = 1, \ldots, |C_v|\}$,

(6) average file streaming rate r,

(7) degree of skew in the file access probability distribution for each client site,

(8) ratio of the cost of one unit of server bandwidth to the cost of one unit of network bandwidth over a hop, i.e., γ, and

(9) the shortest (network) distances between the client sites.

This list omits the more detailed parameters that define the network topology (nodes, arcs, arc weights), the ratio of the average cost of one proxy p stream to the average cost of one origin stream (β_p), the minimum bandwidth available at each client,and the specific distributions of media file streaming rate and access frequencies. We point out that the degree of dispersion among the client sites, determined, for instance, by the shortest distances[1] between them, has a much stronger impact on the optimal CDN solution, i.e., optimal server placement and routing, than the exact network topology connecting clients and access points for server placement. Therefore, in chapter 7, for evaluating optimal server placement and routing solutions, we will use different topologies, with different degrees of client dispersion to evaluate its impact on the optimal CDN design. The cost ratios β_p can be derived from the ratio γ and the placement of the servers. Therefore, for determining the optimal server content for scalable CDNs, assuming fixed server placement and routing, which is the focus of chapters 6, the impact of the network topology (i.e., the distances between the clients) and the cost ratio γ is captured, to a first order, by the parameters β_p, the ratio

[1] The degree of client dispersion can be measured by the maximum or the average distance between any pair of client sites in the network topology.

of the average cost of a proxy p stream to the average cost of one origin stream. The impact of the client bandwidth is captured in the average file streaming rate r.

The principal impact of the distributions of media file streaming rate and access frequencies is captured in the degree of skew in the file access probabilities. In other words, alternative detailed distributions impact the total concurrent bandwidth required to deliver each file, which is captured instead by varying the skew in the file access frequencies. The skew can be varied by varying the parameter θ_c of the Zipf-like distribution of access frequencies for each client site c.

We note further that while the distribution of file durations, $\{T_i, i = 1,\ldots, n\}$, has an impact on the performance of a CDN design model or algorithm, there are potentially infinitely many such distributions to be explored. The principal impact of this distribution is to define the variation in space requirements among the files that have different access frequencies. In other words, the CDN performance may be impacted by whether the most popular files are smaller or larger than the files with intermediate or lower popularity. One might attempt to capture these effects, for example, by defining three classes of file length (e.g., “small”, “medium” and “large”) and by varying the relative popularity of each class. Even in this case, the number of parameters to be varied increases the complexity of exploring the design space considerably. In this thesis, with the purpose of gaining initial insights into various minimum cost CDN solutions, we make the simplifying assumption that all of the media files have same duration T, unless otherwise stated. Further refinement of these insights for varying file durations is left for future research.

3.4.2 Relationships Among the Primary CDN Parameters

In this section, we make the key observation that not all the primary parameters, identified in the previous section, need to be varied independently. For instance, the request arrival rates from each

Table 2: Disk Bandwidth for Particular Disk Models

Disk	Disk Space (GB)	File Streaming Rate	Disk Space (hours of video)	Disk Bandwidth (# of concurrent streams)
Seagate ST15150W	4	1.5 Mb/s	6.36	25
Ultrastar 72ZX	73.4	1.5 Mb/s	116.76	108
Ultrastar 72ZX	73.4	4 Mb/s	43.8	42

client site c, λ_c, and the file duration T affect the required streaming bandwidth for the proxy server that handles those requests only by their product $N_c = \lambda_c \times T$. As an example, if requests from a client site for a 30 minute video (i.e., T = 30 minutes) arrive at the rate of sixty per hour (i.e., λ_c = 60 requests/hour), the placement and content of the proxies is the same as when requests for a 90 minute video arrive at the rate of twenty per hour (if all other parameters are fixed). In both cases, N_c = 30. Therefore, to evaluate alternative CDN solutions, one needs to vary only $N_c = \lambda_c \times T$, instead of λ_c and T.

Similarly, the available proxy disk space P_{space_p} and proxy disk bandwidth P_{bw_p} should not be varied independently as they are constrained by disk technology. For instance, one cannot have a CDN where P_{space_p} = 100 GB and P_{bw_p} = 25 MB/s. Here, we make the assumption, carried out in the rest of this thesis, that the proxy server configuration is balanced for streaming and the available proxy disk bandwidth determines the proxy streaming P_{bw_p}. However, it may be replaced with the proxy i/o network bandwidth, if this happens to be the bottleneck.

Realistic examples of the space and bandwidth available for two disk technologies are given in Table 2. For example, a Seagate Barracuda model ST15150W disk has a capacity of 4GB and a peak transfer rate that varies from 4.8 to 7.32 MB/s. In contrast, the Ultrastar 72ZX, a two-generation more recent disk, released in early 2000, has roughly 18 times the capacity of the Barracuda and roughly 5 times more bandwidth [113]. These two disks illustrate how rapidly disk

technology is evolving. More generally, disk capacity increases by 60% each year while transfer rate increases by only 40% a year [114]. These multiplicative factors indicate that as disk technology evolves, the ratio of bandwidth to storage space available at a server, which is the disk parameter that primarily impacts a CDN solution, decreases. Given the particular disks that are used in a server and the average file streaming rate r, the average number of concurrent streams each server disk can support is computed using the calculations in [108]. The results for such calculations for the two disk technologies are shown in Table 2.

There is also no need to independently vary P_{bw_p}, the average streaming rate r, the product $N_c = \lambda_c \times T$ and the degree of skew in the file access probability distribution (i.e., the parameter θ_c). For instance, the CDN solution is the same whether P_{bw_p} = 8MB/s and r = 1.5 Mbps or P_{bw_p} = 16MB/s and r = 3Mbps. Instead, the average streaming rate r can be defined as a unit (i.e., r = 1) and the proxy bandwidth (in units of the average streaming rate) can be expressed as a fraction B of the total number of streams needed for the delivery of all files, which depends on N_c and θ_c, for all client sites c. In this case, alternative CDN solutions can be evaluated by varying only N_c and θ_c. Alternatively, for evaluating unicast caching strategies, the proxy bandwidth can be expressed as the fraction of the total number of streams needed to deliver the most popular files that fit in the proxy storage space, which depends not only on N_c and θ_c but also on the cache space. In this case, the proxy disk bandwidth is expressed as a fraction of the bandwidth required for delivering the optimal proxy content. The normalizations applied to proxy disk bandwidth for evaluating unicast and scalable CDNs solutions will be further discussed in chapters 5 and 6 of this thesis.

3.4.3 Derived CDN Design Parameters

Given the relationships among the primary parameters, described in the previous section, we identify a small set of derived parameters, which, together, capture the primary impact of the more

Table 3: Derived System and Workload Parameters

System Parameters	**Definition**
C_p	Proxy p cache size, as a fraction of the sum of all n media file sizes
B_p	Proxy p disk bandwidth, as a fraction of the bandwidth needed to deliver all files or the most popular files that can fit in the proxy storage space
β_p	Ratio of the average cost of one proxy p stream to the average cost of one origin stream
Workload Parameters	**Definition**
n	Number of media files (or objects)
N_c	Request arrival rate from client site c, in arrivals per T ($N_c=\lambda_c T$)
θ_c	Parameter of the Zipf-like distribution of file access probability for client site c

basic parameters. These derived parameters, given in Table 3, should be varied over a large range of values in order to evaluate the quality of alternative CDNs over a large design space. Note that Table 3 includes the cost ratios β_p, primary parameters for determining the optimal server content for scalable CDNs that can be derived from the network topology and from the cost ratio γ. However, we point out that for evaluating server placement and routing solutions (in chapter 7), we will use the more basic parameters (i.e., the network topology and γ), instead of β_p.

Chapter 4

Analysis of Educational Media Server Workloads

In spite of the growing popularity of streaming media files [2, 152], there are only a few recent attempts towards analyzing streaming media workloads [4, 42, 44, 78, 79, 103, 104, 109, 142], many of which are parallel to this thesis. However, most of them focus on a few characteristics and, thus, do not provide a very complete characterization. Thus, there is a need for a more thorough characterization of media workloads to provide a better understanding of client load, which is important for designing efficient streaming CDNs and for driving the development of new streaming applications.

This chapter presents a much more complete analysis of server log data for two streaming media servers delivering educational content in two major public universities in the United States. The goals of this study are (1) to provide data for creating synthetic client workloads and (2) to gain insights into the design of efficient CDNs. To gain insights into CDN design, we analyze a number of new measures, including (a) request rate as a function of time, (b) arrival process during periods of stable arrival rate, (c) changes in relative file popularity from day to day and from an hour to the next, (d) distribution of file access frequency during each day, (e) distribution of file segment access frequency and (f) the fraction of accesses to cold objects, i.e., objects that are accessed only very sporadically. We also provide example profiles of client interactive behavior

including the range of data (start and end positions in the file) retrieved by each client request, which can be used to generate realistic client interactive access models.

Most previous workload characterizations analyze data collected over periods where the characteristic being analyzed is likely to change significantly. Such characterizations may result in misleading conclusions, as the distribution obtained is the result of a combination of several distributions. One key contribution of this work is to emphasize the importance of analyzing the time varying aspect of the workload by splitting the characterization process into two phases. We first identify the periods during which the aspect being analyzed remains approximately stationary and then we separately analyze the data collected for each such period. This approach, pioneered in our recent paper [10] and followed in two recent studies [42, 142], allowed us to obtain meaningful distributions and gain insights into how characteristics such as request rate or relative file popularity change over time.

The remainder of this chapter is organized as follows. Section 4.1 provides some background on the two media servers. Sections 4.2 and 4.3 analyze the general load and file access pattern on each server, respectively. Section 4.4 analyzes client interactivity, including session characteristics, and the server bandwidth savings from multicast delivery. Section 4.5 summarizes the main findings of this work. Additional workload data is provided in appendix A.

4.1 Server Logs

This section describes the two media servers considered. Both servers deliver educational content in two major public universities in the United States and contain higher quality videos than servers or client workloads previously analyzed [4, 44, 78, 79, 103, 109]. The tools used to analyze the

server logs include (a) a software[2] that analyzes media server logs in either Real Server or Windows Media Server format by creating a common internal representation of the log fields and provides statistics on several aspects of the workload and (b) a set of scripts we created for extracting and analyzing specific aspects of the workload based on the log internal representation created by the software.

4.1.1 The eTeach Server

The *eTeach* server [64] was put into production in 2000 to deliver the content for a computer science undergraduate course (CS 310) at the University of Wisconsin-Madison with an enrollment of 280 students. There are no live lectures for that course. Instead, video lectures and laboratory demos are archived at the server and later viewed by the students. Thus, analysis of the eTeach data enables us to look at client requests for a particular course that is only available on the server.

The eTeach videos are recorded in Windows Media Server [151] format. Students access the videos using a version of the Windows Media Player embedded in a customized web browser window. The browser displays the video stream in one frame, the outline for the lecture in another frame and the slides that go along with the lecture in a third frame. When a client requests a video, the browser starts playing the beginning of it and displays the outline and the first slide. The eTeach interface allows clients to pause, resume and skip to markers in the video, corresponding to specific topics in the outline. Fast forwarding and rewinding were not provided by the interface by the time the logs were collected (Fall 2000 semester). However, they were available if the clients accessed the Windows Media Player directly.

---

[2] The software was written by Jeffrey Krueger, an undergraduate student at University of Wisconsin-Madison

The eTeach server records a separate log entry for each interactive request, i.e., every time a client issues a request to start, pause, resume or skip to a marker, a new log entry for that request is created. Furthermore, it also explicitly registers the starting point, in the video, of each request.

4.1.2 The BIBS Server

The BIBS, *Berkeley Internet Broadcasting System*, server [23] has been in operation since January 1998 and serves lectures for several courses across a number of different fields including art, biology, chemistry, computer science and electrical engineering. During the Spring 2000 period, covered by our logs, lectures from eleven different courses were available on the server. Unlike the lectures in the eTeach system, the lectures available at the BIBS server are always provided live in the classroom first and made available online for students who missed them or who want to review the material. In a few cases, the lectures are also available live on the server.

The BIBS lectures are encoded in Real Server format and are delivered live or on-demand using the BIBS RealServer G2 server [119]. Students view most of the lectures using an ordinary Real Player application, which provides pausing, resuming, fast-forwarding and rewinding but no markers to jump to. Unlike eTeach, BIBS records only one entry per session including all pauses, resumes, fast forwards and rewinds within the video. The duration and bytes of media sent to the client, registered in the log, includes all the media sent during the session and, thus, may not be a contiguous segment of video. Moreover, the start position of the session and information about each interactive request within a session are optionally recorded but are not present in our logs. Thus, BIBS data is analyzed for overall server load and file access characteristics. Client interactivity and file segment characteristics are analyzed only on eTeach.

Table 4: Summary of Characteristics of Server Logs

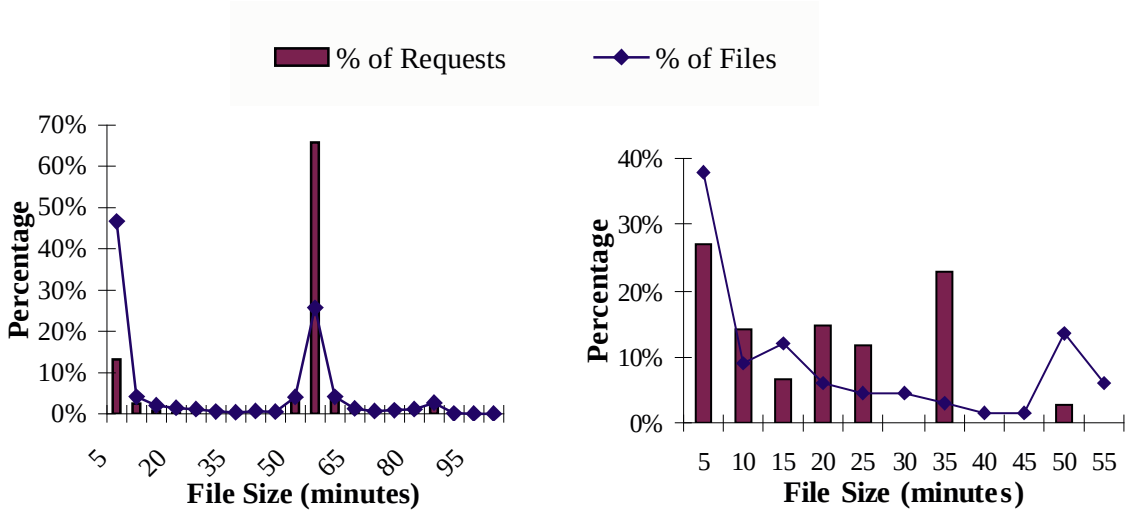
Item	BIBS	eTeach
Log Duration (days)	02/01/00 – 05/19/00 (109)	09/12/00 – 10/12/00 (31)
Total Number of Stored Files Accessed	1350 files	73 files
Number of Stored Files Accessed Only Once	110 files	12 files
Number of Requests for Stored (Live) Content	66,694 sessions (22,411)	17,233 requests
Number of Hours of Stored Content Delivered	19,518 hours	475 hours
Average (CoV) Number of Stored Requests Sent per Day	606 sessions/day (0.27)	538 requests/day (0.86)
Average (CoV) Number of Stored Hours Sent per Day	177 hours/day (0.61)	15 hours/day (0.81)
Average (CoV) Number of Stored Files Requested per Day	157 files/day (0.06)	19 files/day (0.13)

4.2 Server Load Characteristics

This section characterizes several aspects of the load on each server, including file sizes and bit rates (section 4.2.1), daily and hourly load variation (section 4.2.2), request arrival process (section 4.2.3) and media delivered per session (section 4.2.3). An overview of the logs and the overall load on each server during the time covered by the logs are shown in Table 4. The eTeach logs end the day after the first course exam and the BIBS logs end during the final exam week for the semester. For each server, a day is defined to begin at the hour that typically has the lightest load, i.e., 3am for eTeach and 7am for BIBS.

4.2.1 File Characteristics

The files *stored* at each server are grouped based on their sizes (in minutes). Figure 2 shows the distribution of file sizes and the fraction of requests to each file size range on each server. Most of the BIBS files are either small announcements (under 5 minutes), or approximately one-hour lectures. The majority (i.e., 70%) of the requests for *stored* content are for the 50-55 minute



a) BIBS **b) eTeach**

Figure 2: Distribution of File Sizes

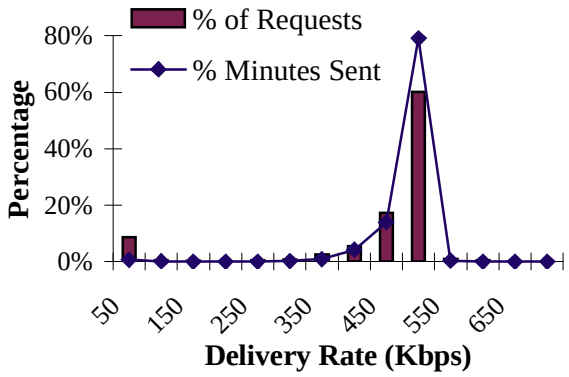
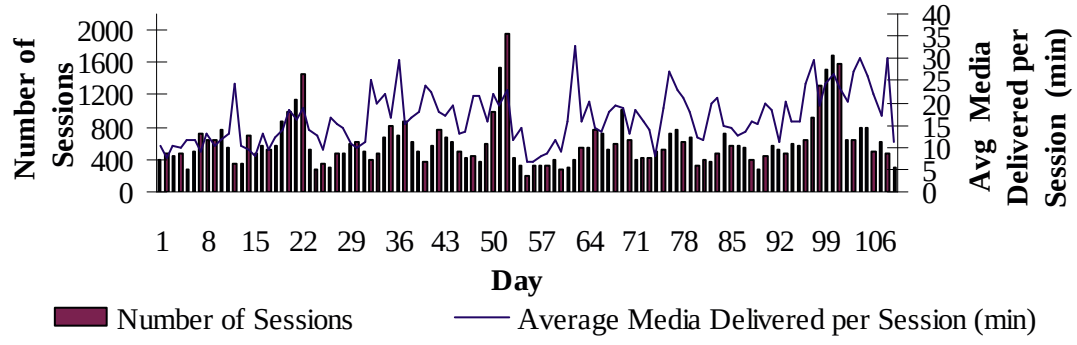
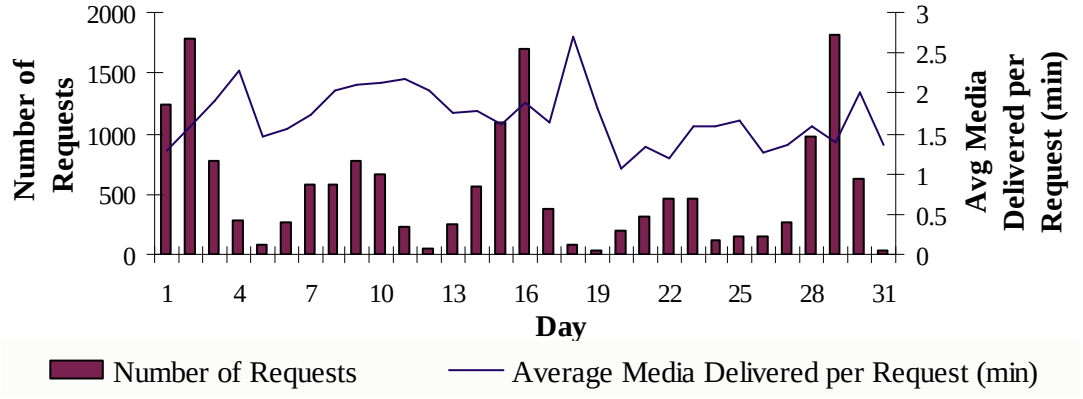


Figure 3: Distribution of Delivery Rate in BIBS

lectures, similar to previously studied server workloads containing stored content from classroom lectures [78, 109] and internal meetings at a large corporation [42]. The eTeach server has a much higher variability in stored and requested file sizes. A large fraction (i.e., 40%) of the stored files is under 5 minutes, but there are also a significant number of files in each of the other categories. The percentage of requests for each category is also very widely distributed. Such a high variability in the distribution of file sizes has been previously observed only in [42]. Nearly all videos requested on eTeach are encoded for 300 kbps. For BIBS, Figure 3 shows that the vast majority of the



a) BIBS



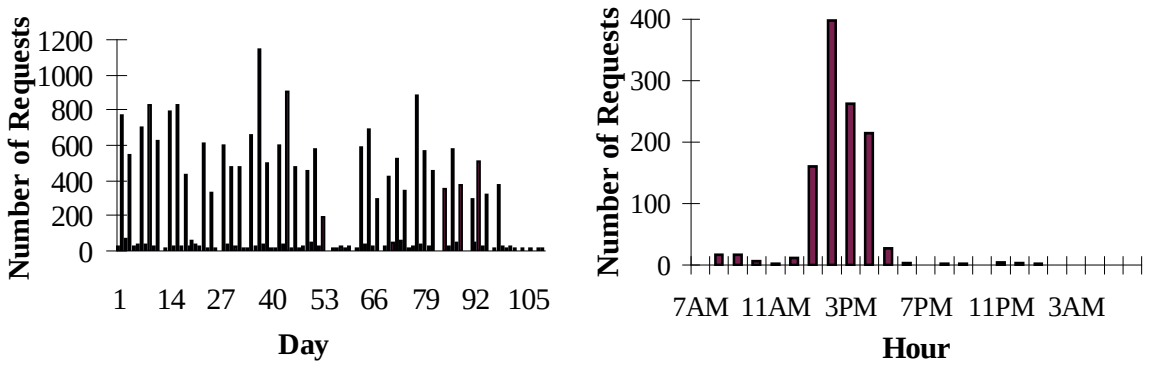
b) eTeach

Figure 4: Server Load per Day for Stored Content

requests and the media delivered are for videos encoded at 350-500 kbps. These delivery rates are considerably higher than the rates observed in previous streaming workload studies [4, 42, 44, 78, 79, 109].

4.2.2 Daily and Hourly Server Load

The daily load fluctuations for stored content on each server are shown in Figure 4. The curves have the bell shaped profile, also previously observed in [44, 78, 79, 104, 142], with peaks of activity during weekdays and lighter loads on the weekends, especially Saturdays. Typically, BIBS has 500-800 sessions per day and 15-25 minutes of media delivered per session and eTeach has



a) Live Requests per Day **b) Example Live Requests per Hour**

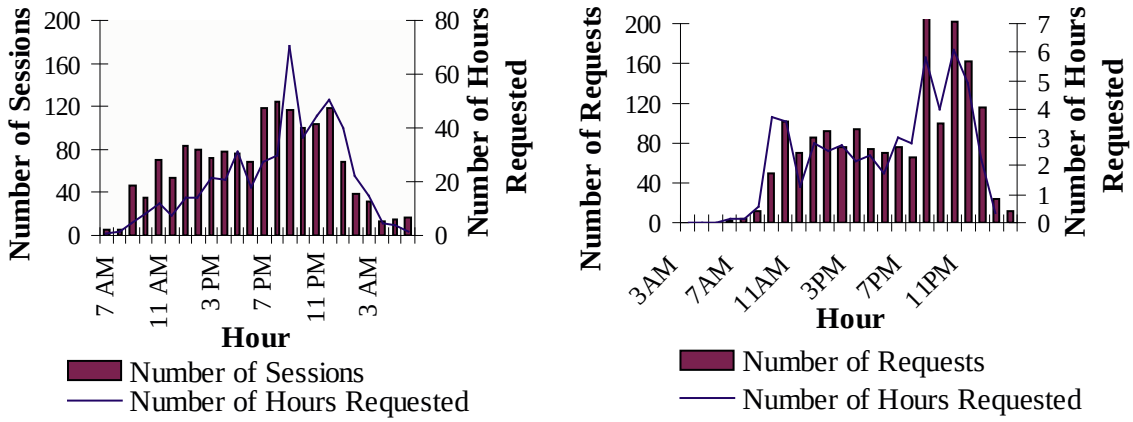
Figure 5: Server Load for Live Content in BIBS

Table 5: Selected High Load Days

Server	**Maximum # requests**	**Maximum # hours sent**	**Maximum # mins sent / # requests**	**Minimum # mins sent / # requests**
eTeach	October 10th (1812)	September 27th (54 hours)	September 15th (2.3 minutes / request)	October 3rd (1.2 minutes / request)
BIBS	March 22nd (1540)	March 23rd (755 hours)	May 14th (30 minutes / session)	February 2nd (7.6 minutes / session)

500-1800 requests per day and a very short average stream duration of 1.5-2 minutes, reflecting a highly interactive client community. One measure that is very similar on both servers and do not vary much from day to day or hour to hour is the coefficient of variation in the number of minutes of media delivered per request or session (1.5-2.5). Figure 5 shows the daily and typically hourly number of request arrivals for live content at the BIBS server. The average number of concurrent viewers for live streams typically vary from 15-35. Request for live content are not analyzed further as this work focuses on characterizing access to *stored* content.

Based on the daily usage statistics, representative days of high load activity are selected for more deep characterization. The selected days are shown in Table 5. Approximately ten other high



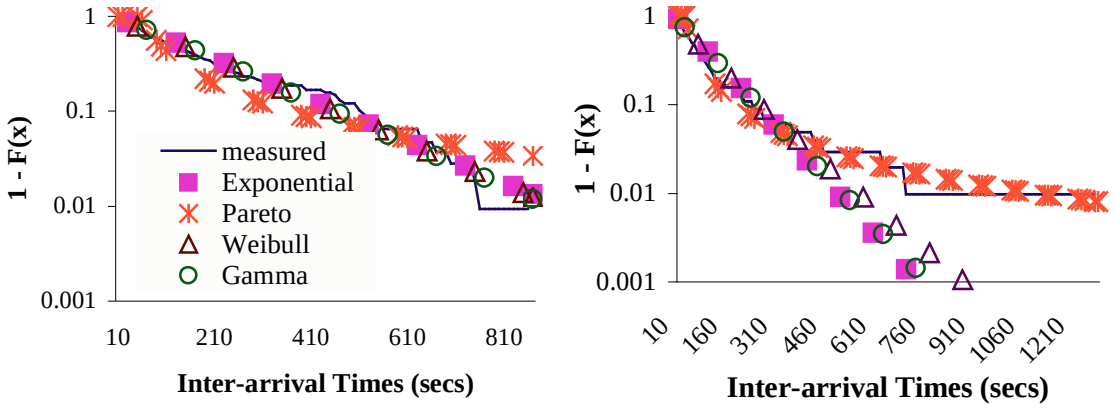
a) BIBS, March 22nd **b) eTeach, September 27th**

Figure 6: Server Load per Hour for Stored Content

load days are also used in the analysis provided in this chapter. Figure 6 shows a typical profile of request (or session) arrivals per hour during one of these days for stored content at each server. Profiles like these are created for many different days in order to find periods when the arrival rate is approximately stationary. Examples of such periods are from 11am to 6pm on March 22nd in BIBS (Figure 6(a)) and from 11am to 7pm on September 27th, in eTeach (Figure 6(b)). These periods are used in the next section for the characterization of request (or session) inter-arrival times. Profiles like the one in Figure 6 are also useful to determine how frequently request rate changes over time, which is important to determine how frequently to update a CDN configuration. In both servers, we found a number of long periods of request rate stability with duration from 7 to 10 hours, but we also found some relatively short periods of rate stability (e.g., a couple of hours)

4.2.3 Request Inter-arrival Times

For all periods of stationary overall request arrival rate identified on each server as well as for each *file* that had a sufficient number of requests during the period, the method of moments is used to fit exponential, gamma, Weibull and Pareto distributions to determine the distribution of time between



a) BIBS, March 23rd, 6pm – 11pm **b) eTeach, October 10th, 10am – noon**

Figure 7: Example Distributions of Inter-Arrival Times

Table 6: Distribution of Inter-Arrival Times

Server	Overall			Per File		
	Most Observed Distribution	Mean (seconds), CV		Most Observed Distribution	Mean (seconds), CV	
		Min Mean, CV	Max Mean, CV		Min Mean, CV	Max Mean, CV
BIBS	Exponential	33, 0.87	104, 1.05	Exponential	183, 0.99	516, 1.28
eTeach	Pareto	43, 0.98	68, 1.82	Pareto	57, 0.97	351, 2.4

requests. Examples of fitted curves are shown in Figure 7. Like other types of web workloads [18, 112, 150], the time between session arrivals in BIBS is best modeled by an exponential distribution. On the other hand, in eTeach, the time between request arrivals usually has a heavier tailed distribution, better modeled with a Pareto distribution with shape parameter α varying from 1.02 to 1.98 and from 0.92 to 1.49 for the distributions of overall and per-file inter-arrival times, respectively. The probability density function of a Pareto distribution for a variable X is given by $f(x) = \alpha k^{\alpha}/x^{(\alpha+1)}$, where α is the shape paramenter and k is the smallest value for X.

The range of the means and coefficient of variation (CV) for the inter-arrival times, overall and per file, for each server are shown in Table 6. Note that although the distribution of inter-

arrival times for requests in eTeach is better modeled with a Pareto distribution, the coefficient of variations of the distributions observed are much higher than the coefficient of variation of a Poisson distribution (i.e., CV = 1). In fact, the average coefficient of variations over all days analyzed, for the overall and per-file distribution of inter-arrival times in eTeach, is on average only 1.35. This result implies that the assumption of Poisson request arrivals may not be too innacurate for eTeach request arrivals.

To our knowledge, such distribution has only been successfully characterized in one previous work [103] and one subsequent work [142]. The attempt to characterize inter-arrival times in [4] resulted in no clear indication of a distribution, but the authors analyze data collected over six months with no attempt to break it into periods of stationary arrival rate. Our results are consistent with a previous characterization of real audio traffic emanating from a popular Internet audio service [103]. On the other hand, both session and request inter-arrival times to a live popular "reality TV show" are found to be exponentially distributed in [142].

4.2.4 Media Delivered per Session

A significant fraction of requests on both servers has very short duration, as also observed in other recent studies [4, 42, 44, 78, 79, 104, 109]. In fact, 80-90% of the eTeach daily interactive requests and 30-70% of the BIBS daily sessions request less than 3 minutes of media. This section characterizes the media delivered per session on the BIBS server. Interactive requests and sessions in eTeach are analyzed in section 4.4.

As illustrated in Figure 6(a), the media delivered per session fluctuates along the day. In order to characterize the distribution of media delivered per session, which may depend on the duration of the requested file, we have identified several periods of approximately stationary average

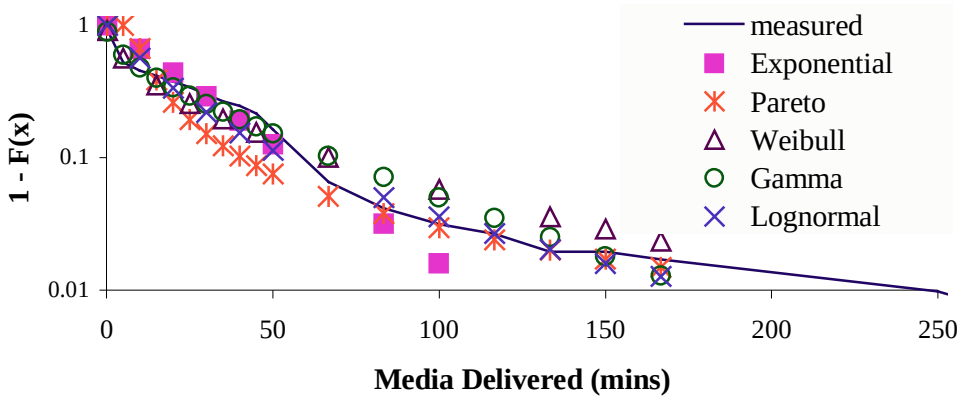


Figure 8: Example Distribution of Media Delivered per Session

(BIBS, May 14th, 11am-10pm, File Size: 50-55 minutes)

Table 7: Distribution of Media Delivered per Session in BIBS

File Size (mins)	Distribution		Mean (minutes), CV	
	Most Observed	Other	Min Mean, CV	Max Mean, CV
0-5	Lognormal	Exponential	0.74, 1.1	1.5, 1.64
50-55	Gamma + Pareto	Gamma + Lognormal	20, 1.13	26, 2.35

number of minutes per session for each of the most popular file length classes, i.e., 0-5 and 50-55 minute files. The measured distribution in each such period is fitted against several distributions, as illustrated in Figure 8. Table 7 summarizes the distribution that provides the best fit for each class of file sizes. Not surprisingly, the distribution does depend on the file size. For short videos, a single distribution, most commonly lognormal is the best fit for the data. For long videos, a hybrid distribution, which combines a gamma distribution for the body of the curve and Pareto for the tail, has a better fit. Table 7 also provides other distributions that were observed for each file length class, as well as a summary of the observed minimum and maximum means, and minimum and maximum coefficient of variations (CV), of the distributions that occurred during the periods of stationary media delivered per session for each file length range. During other periods, the average

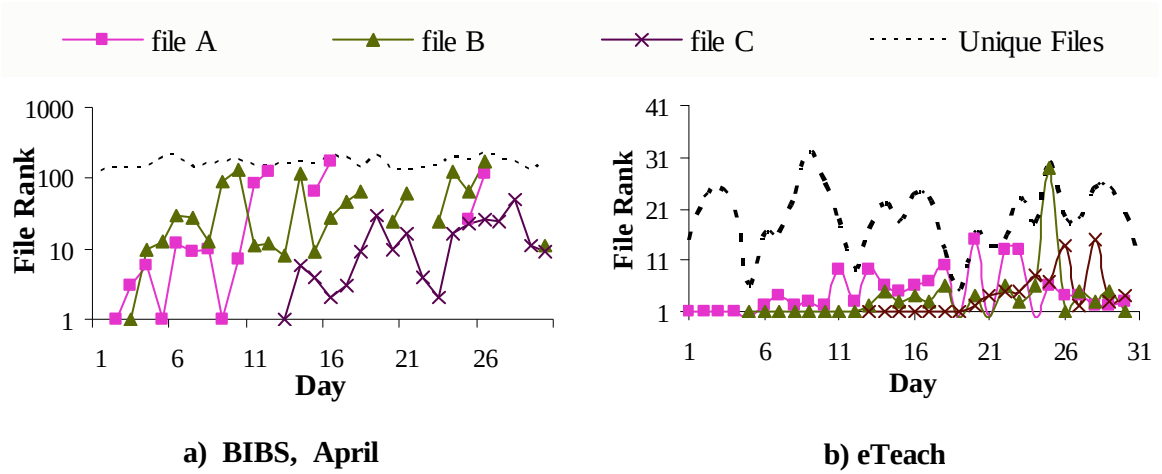
number of minutes of media delivered per session was typically in the range of 0.5 to 2.5 for videos under five minutes and 10 to 30 minutes for the 50-55 minute videos.

Our results are consistent with previous works that show that the distribution of media delivered per session is heavy tailed [44, 103]. This distribution was characterized as either gamma or lognormal for sessions to 70 minute long lectures in the MANIC system [109] and as lognormal for sessions delivering live content from a reality show [142].

4.3 File Access Characteristics

To the best of our knowledge, previous attempts towards analyzing file access patterns in streaming media workloads focus mainly on the distribution of file access frequency. We provide a much more complete characterization, including daily and hourly variations in relative file access frequency, file and file segment access frequencies and temporal distribution of accesses to infrequently requested files and segments to provide insights into the design of efficient media caching strategies. Some of the unique aspects analyzed in this work are also analyzed in a more recent work [42].

We also analyzed temporal locality in the client requests for files (BIBS and eTeach) and file segments (eTeach) using the distribution of *stack distances* [131], which measures the number of intervening references between references to the same file. This method has been used to characterize the temporal locality for traditional web workloads [12, 17, 22, 48, 86] and for a media workload [4]. We found that temporal locality seems to vary significantly across different days on each server. Results for the temporal locality in the accesses to file segments in eTeach are shown in appendix A.



a) BIBS, April **b) eTeach**

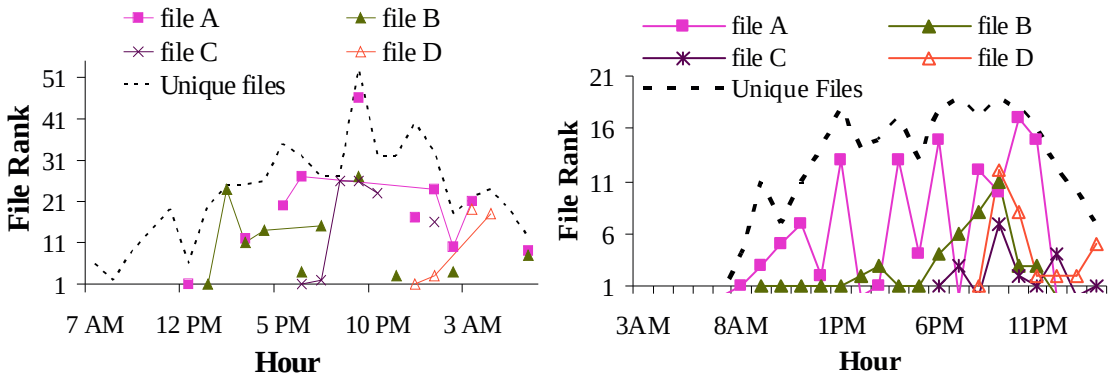
Figure 9: Example Daily Evolution of File Popularity

4.3.1 Daily and Hourly Shifts in File Popularity

Both the eTeach and BIBS logs used in this study span a reasonably long period of time (one-three months), during which file access frequency may change significantly. This section analyzes how relative file popularity changes over time by considering the plots of daily and hourly access frequency rank of various files over many different periods for each server.

For each BIBS and eTeach file that has the highest number of requests on a given day, Figures 9(a) and 9(b) plot the ranking of that file on each subsequent day. These plots illustrate that, typically, the relative popularity of each file fluctuates greatly from day to day (especially in BIBS) and in different ways for different files. In eTeach, the most popular file tends to remain in the same rank for roughly one week, and then it has lower popularity until a few days before the exam on October 12th.

Figure 10(a) shows that relative file popularity changes significantly also during a single day, on a per hour basis on the BIBS server. During the days before the course exam, when students are probably reviewing all the material offered previously in the semester, file popularity on the



a) BIBS, May 14th **b) eTeach, October 10th**

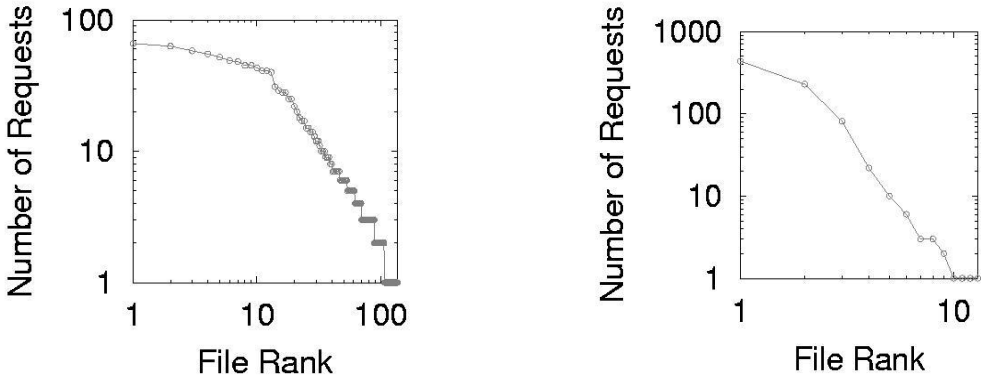
Figure 10: Example Hourly Evolution of File Popularity

eTeach server fluctuates hourly almost as much as shown for the BIBS server, as shown in Figure 10(b). On other days, a particular file, probably the most recent lecture, remains the highest ranked file over most hours in the day (as shown in appendix A). Further examples of hourly evolution of relative file popularity in eTeach and in BIBS are shown in appendix A

These results suggest that a CDN configuration (e.g. server content) may need to be updated to reflect the variations on file popularity. A topic for future research suggested by this data and relevant to CDN design is whether analysis of historical information on the variations of access frequency during a day for a given file might be used to accurately predict the variation on its access frequency on the following day. This data also suggests that static broadcast of popular files [83] may not work well in these environments.

4.3.2 Distribution of File Access Frequency

The distribution of file access frequency should be analyzed over periods when it is roughly stable. However, since file popularity fluctuates from day to day and from hour to hour, only a few such periods are found on either server over all the days analyzed. Therefore, in addition to characterizing the distribution of file access frequencies during each such period, the distributions

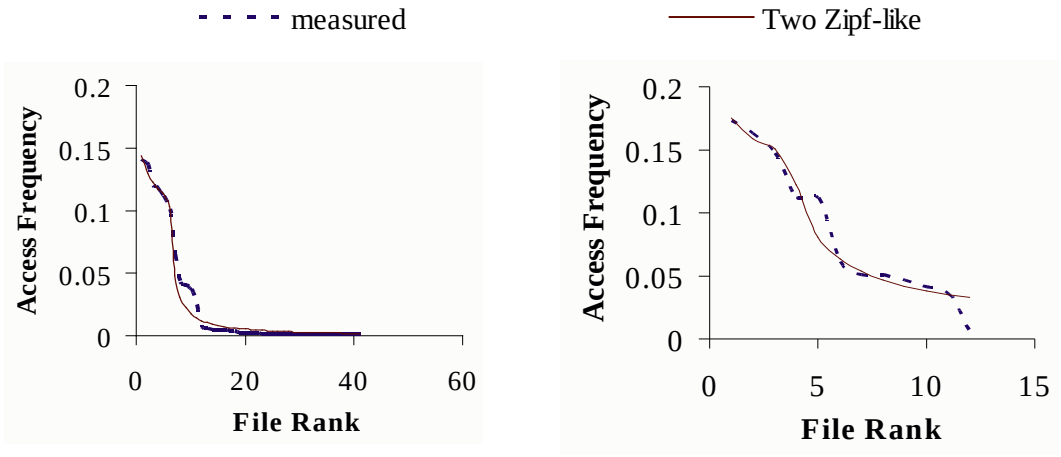


a) BIBS, May 9th, File Size: 50-55 mins **b) eTeach, September 27th,File Size: 0-5 mins**

Figure 11: Observed Distributions of File Access Frequencies (log-log plots)

for many individual days are also analyzed, recognizing that further research is needed to characterize how the file accesses are distributed throughout the day. We chose to analyze file access frequency per day because for shorter periods, the number of accesses per file in either server is too small to guarantee statistical meaning of the distribution. In order to facilitate the generation of synthetic workload, file access frequency is also analyzed separately for each popular file size range on each server.

Figure 11 shows the log-log plot of the number of accesses versus file rank for the files of a specific duration accessed during a day on BIBS and eTeach servers. Figure 11(a) shows that the curve is approximately linear for the first thirteen files and it is also approximately linear (with a different slope) for the remaining files. Similarly, the curve in Figure 11(b) has also two distinct linear regions. Note that a straight line on a log-log plot of the number of accesses versus file rank indicates a Zipf-like distribution for the access frequency. In general, for all the days and periods of table file popularity analyzed, the distribution of access frequencies in the log-log plot has two distinct approximately linear regions. The two linear regions are less well defined for eTeach, perhaps due to the smaller number of unique files accessed per day. Therefore, the distribution of



a) BIBS (March 23rd, 6pm-1am, File Size: 50-55min)

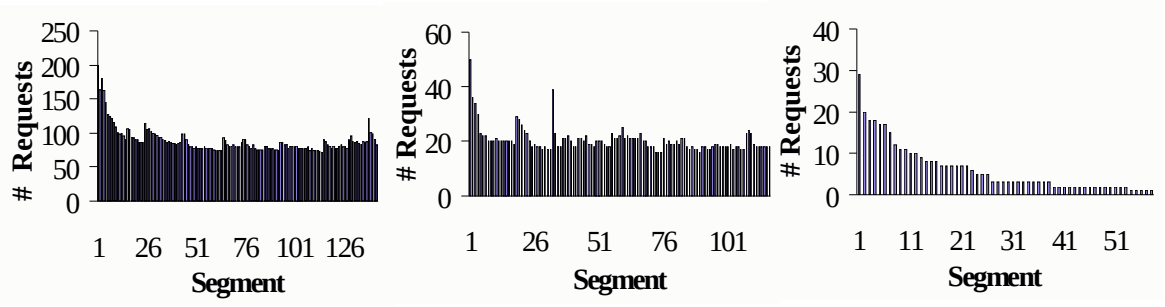
b) eTeach (October 11th, 0am-11pm, File Size: 0-5min)

Figure 12: Example Distributions of File Access Frequencies

access frequencies to *all* media files on *both* servers or to files of a given duration can be very accurately approximated by the *concatenation* of *two* Zipf-like distributions[3], one for each linear region, as illustrated in Figure 12. The number of files, the total access probability and the Zipf parameter θ in each region depend on the period analyzed and on the file size. Typical values for these parameters observed in the workloads are provided in appendix A.

This contrasts with previous characterizations of access frequencies for Web documents [12, 22, 25], videos at a video rental store [52] and other streaming workloads [42, 44, 142], which model file access frequencies using only one Zipf distribution. However, some of these works ignore some of the most accessed files, over which the slope of the curve is flatter in the log-log plot [25, 42]. The workload analyzed in [25] includes millions of web documents and the top hundred files ignored correspond to only a tiny fraction of all files. On the other hand, the analysis

[3] Recall that, in a Zipf-like distribution, Prob(access file i) = C/i^{θ}, where $\theta > 0$.



a) Most Accessed File **b) 4th Most Accessed File** **c) 6th Most Accessed File**

Figure 13: Example Distributions of Accesses to 10-Second File Segments in eTeach (September 27th)

in [42] includes a number of files in the order of hundreds. In that case, by ignoring the 10-20 most frequently requested files, the authors may have ignored a significant portion of the actual distribution. Given the relatively small number of files accessed on each day on BIBS and eTeach, we do not ignore any file and model file access frequencies using two Zipf-like distributions.

4.3.3 Segment Access Frequency Distribution

Given that most requests in eTeach and sessions in BIBS are for only a small amount of media (as discussed in section 4.2.4), the next step towards a thorough characterization of file access frequency is to analyze the accesses to different segments of the same file. In addition to providing basis for generating more realistic synthetic workloads, this analysis also provides insights into the design of file caching strategies. As noted in section 4.1.2, information on per-segment file access is not available on the BIBS logs used in this work. Thus, this section analyzes file segment access frequency only in eTeach.

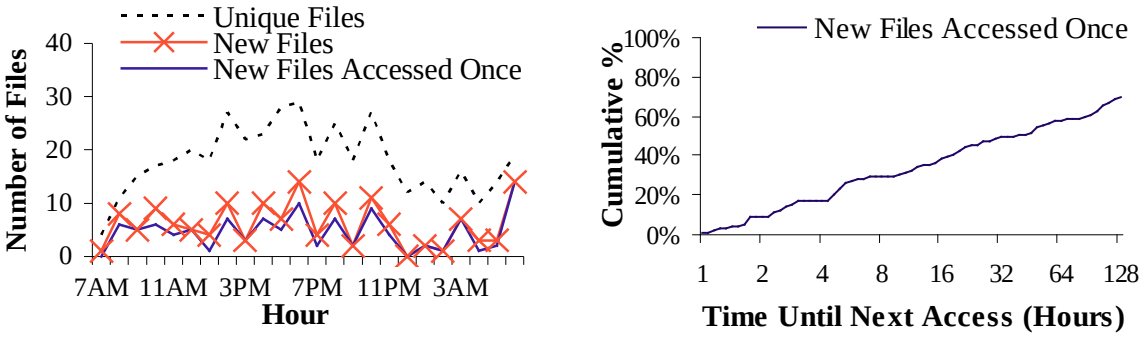
Figure 13 shows the observed number of accesses to each ten-second segment of three different files, with different relative file access frequencies, on a given day in eTeach. One interesting result, illustrated in Figures 13(a) and 13(b) is that the distribution of access frequency is roughly uniform for all segments of the eTeach media files with moderately to high relative

access frequency, in spite of the large fraction of requests of very short duration. On the other hand, for the media files that have the lowest access frequency (e.g., below the median for each day analyzed), the distribution of segment access frequency is skewed towards the earlier segments, as illustrated in Figure 13(c). The distributions of file segment access frequency for files accessed on other days in eTeach are shown in appendix A. The overall same conclusions were observed for all days analyzed, with the skew of the distribution for less frequently accessed files varying depending on the day analyzed.

To the best of our knowledge, this is the only work that analyzes segment access frequency. One key question is whether our conclusions hold for other media servers. One important implication from these observations is that it might not be necessary to keep information on the access frequency of each file segment, separately. In practice, one could measure the access frequency for only few segments and then fit a curve to estimate the skew of the distribution, if there is one. Furthermore, though most individual client requests are for only small amount of media, for unicast CDNs, it might be more cost-effective to *fully* cache the most accessed files, instead of independently caching different file segments.

4.3.4 Fraction of Accesses to Cold Objects

One consequence of the Zipf-like distribution of file access frequency is the large number of files that are requested only a few times. In fact, for the one-week long workload analyzed in [44], 78% of the media objects are requested only once during the week. For the workloads of the two HP servers analyzed in [42], on average 18% of the files are accessed only once during 1.75-2.5 years. Despite the large fraction of files that are requested only a few times, many previously proposed caching strategies cache an object when it is requested even without any prior knowledge of its access frequency. Whereas caching small unpopular web files should not affect the performance



a) Number of File Accesses per Hour **b) Distribution of Time Until Next Access**

Figure 14: Accesses to New Files Each Hour (BIBS, March 23rd, h = 4)

significantly, caching large and bandwidth intensive media objects that are accessed only sporadically, may incur significant write overhead if the object is not accessed at least once before it is evicted from the cache. This section provides some insight into whether this is likely to be an issue in the interactive media servers studied.

For the following analysis, *new* media files (or *new* media segments) accessed in a given hour are defined to be the files (or file segments) that were not accessed in the previous h hours, where h = 1, 2, 4 or 8. Those files are not likely to be in the cache by the time they are requested. Figure 14(a) shows for every hour of a given day on the BIBS server, the number of new files accessed and the number of new files accessed only once during that hour, where a *new* file is defined for h = 4. The fraction of new files that are accessed only once is very high (70–100%). Furthermore, Figure 14(b) shows, for the new files that are accessed only once, the distribution of time until the next access. A significant fraction (i.e., 70%) of these files is not accessed again within the next 8 hours. Similar results are obtained for h = 1, 2, 4 and 8, and for every day analyzed on both servers, as summarized in Table 8 for the BIBS server (similar data for eTeach is shown in appendix A). Table 9 shows that the same conclusion holds for accesses to *one-minute* segments within the

Table 8: Summary of Accesses to New Media Files in BIBS

Day	*h*	Median # new	Average # new accessed once / # new	Time Until Next Access			
				% > 4 hr	% > 8 hr	% > 16 hr	% > 32 hr
2/2	2	5	0.70	62	54	48	38
	8	5	0.67	71	64	58	51
3/22	2	10	0.78	71	59	50	40
	8	6	0.81	84	72	67	61
3/23	2	8	0.70	73	64	54	44
	8	4	0.79	82	76	68	57
5/14	2	14	0.81	65	54	39	26
	8	10	0.84	74	64	49	34

Table 9: Summary of Accesses to New Media Segments in eTeach

Day	*h*	Median # new	Average # new accessed once / # new	Time Until Next Access			
				% > 4 hr	% > 8 hr	% > 16 hr	% > 32 hr
9/15	2	10	0.82	75	66	66	59
	8	4	0.82	72	71	71	63
9/27	2	10	0.81	69	68	62	60
	8	8	0.78	80	78	67	64
10/3	2	2	0.70	83	83	79	56
	8	2	0.87	84	84	79	57
10/10	2	12	0.62	58	53	53	52
	8	5	0.7	64	55	55	54

videos on the eTeach server. These results indicate that caching a media object with no information about its popularity may not be a good strategy as it reduces the bandwidth available to deliver files to clients with little benefit in many cases. This motivates the development of more efficient caching strategies for media files.

To the best of our knowledge, no previous characterization of streaming workload has addressed this issue. On the other hand, more recently, Cherkasova *et al* built on our work and found that the number of sessions to new files and the number of bytes of new files delivered by

the HP servers represent a very significant portion of all accesses or bytes delivered [42]. This is in contrast with traditional web workload, where only about 2% of the requests are for newly created content [43].

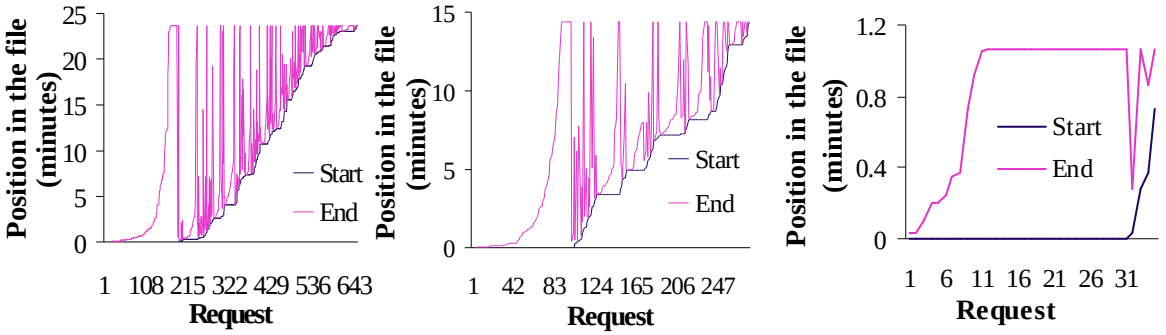
4.4 Client Interactivity

This section provides an analysis of client interactivity in eTeach, where up to 90% of the requests received each day are for fewer than three minutes of video. This analysis provides a set of profiles of client interactive access pattern (section 4.4.2), an evaluation of the performance of multicast streaming protocols in the interactive eTeach environment (section 4.4.3) and a session model for generating synthetic workloads (section 4.4.4).

4.4.1 Client Interactive Access Patterns

This section shows profiles of the client behavior in the eTeach server. These profiles provide useful information for future development of more accurate client interactive access models, efficient prefetching strategies and delivery protocols. In particular, data derived from these profiles is used in the next section to analyze the efficiency of multicast streaming in the eTeach environment and is used in [135] to develop models for client interactivity.

Profiles of the client requests for the most requested file on two given days and for the 3rd most requested file on another day are shown in Figure 15. The plots show the start position and the end position of each request for the file, ordered by increasing start position. For a given point in the *x*-axis, the difference between the two curves is equal to the number of minutes of media delivered. Requests that have the same start position are ordered in increasing duration. The figures illustrate three different patterns of client accesses. In Figure 15(a), there are requests starting at many different arbitrary positions in the file. We conjecture that this is a consequence of clients



a) Most Popular File on September 27th **b) Most Popular File on October 10th** **c) 3rd Most Popular File on October 3rd**

Figure 15: Interactive Client Requests in eTeach

pausing and resuming the playback, probably to take notes, as there are no classroom lectures. In Figure 15(b) there are more clear flat regions in the start curve, which probably correspond to markers in the lecture the students jump to. Furthermore, perhaps these flat regions reflect that, towards the end of the semester (October 10th), the students are using the markers more often as they become more familiar with the content of the video. It is also interesting that in both Figures 15(a) and 15(b), for any start position, there is at least a small fraction of the requests for the rest of the file. Furthermore, though most of the requests are for small portions of the video, there is also a non-negligible fraction of the requests that retrieve the whole file. In Figure 15(c), most of the requests are for the whole file. This access pattern was observed, in many days, for very small files such as the one in Figure 15(c), which is approximate 1 minute long.

4.4.2 Multicast Delivery

Multicast streaming protocols, such as Bandwidth Skimming [61, 62] has been previously evaluated assuming clients request the whole file. The performance of such protocols in the presence of client interactivity is not known. In particular, whether partial accesses significantly

Table 10: Server Load Reduction from Multicast Delivery in eTeach

Period	File Rank	Average Number of Concurrent Streams		
		Unicast	Multicast	Savings
Sep 27th 10PM-11PM	1	3.28	2.43	26%
Sep 27th 11PM	1	2.83	2.02	29%
Sep 27th 10AM-11AM	1	1.73	1.27	26%
Oct 3rd 3PM-4PM	2	0.69	0.59	14%
Oct 10th 10AM-12PM	1	1.48	1.21	18%
Oct 10th 12PM	1	1.74	1.33	23%
Oct 10th 8PM-9PM	5	0.42	0.48	13%

limit the potential for streaming merging and thus the efficiency of multicast streaming is still an open question.

To quantify the server bandwidth savings from using multicast delivery in eTeach, Closest Target, a variation of the Bandwidth Skimming protocol [61, 62], is simulated using client traces collected during periods of stationary total request rate, varying from 10 to 70 requests per hour. Table 10 compares the average number of concurrent streams that the server must support to deliver the most requested files during each such period for unicast and multicast delivery. The results show that, despite the bandwidth savings from multicast vary significantly with the period and the file. However, the bandwidth savings for the traces analyzed can be up to 30%

To better understand why the bandwidth savings from multicast are relatively modest, Figure 16 shows the client requests for the most requested file on a day (September 27th) in eTeach. The start and end positions of each request are plotted in chronological order, i.e., the requests are sorted by their arrival times. The *x*-axis represents time and *y*-axis the position in the file, each measured in minutes. Each line, connected by two points, corresponds to one request: the *x* and *y* values of the lower point represent, respectively, the time when the request arrived and the start position into the file; the *x* and *y* values of the upper point represent the time when the request was

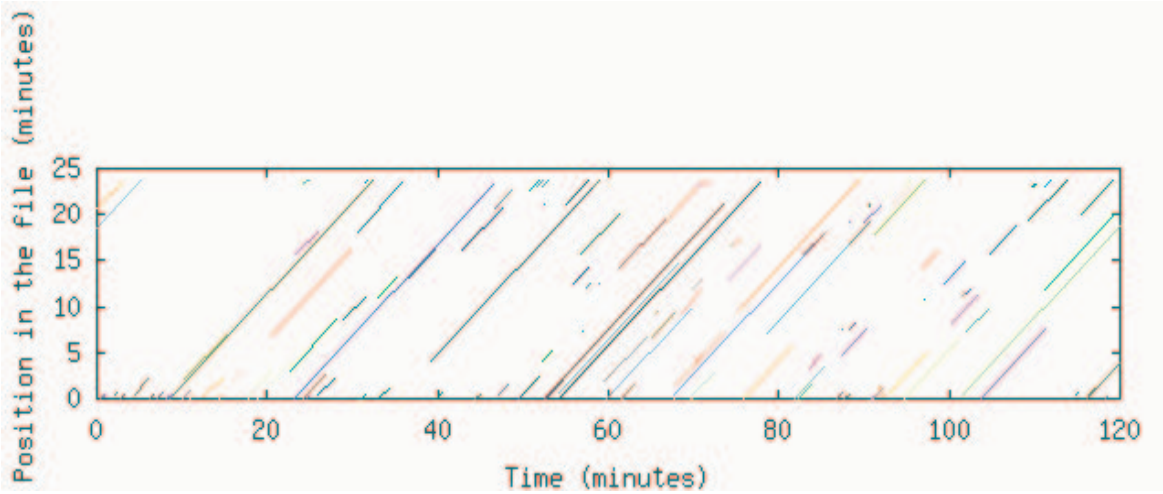


Figure 16: Snapshot of Client Request Arrivals to Most Popular File (eTeach, Sept. 27th)

completed and the end position in the file. For the sake of clarity, Figure 16 shows only a snapshot of a period of 2 hours during the day. Despite the presence of a few long requests and some bursty arrivals, most of the consecutive requests are small and do not overlap, which limits the protocol merging efficiency. In a parallel work, Chesire *et al* show that, for a workload with a large fraction of short streams, two overlapped requests for the same file tend to have a high fraction of overlap [44]. However, since only overlapping requests are considered, it is not clear whether the bandwidth savings from multicast would be significantly different for their workload.

Note that further bandwidth savings might be expected for higher client loads. In fact, Tan *et al* recently developed tight lower bounds on the server bandwidth requirements for interactive workloads and showed that, in the worst case, it increases with the square root of client request rate [135]. Furthermore, bandwidth savings could be improved if, instead of interrupting transmission at each client pause request, the server would continue delivering the content, at least for a certain threshold period. This content could be buffered at the client, or at a proxy server closer to the

client. If the client resumes transmission at the same point in the video, a short time later, the content would be found locally.

4.4.3 Session Characteristics

Client sessions, consisting of alternating media ON and OFF times for a given client accessing a given video, are not identified in eTeach. In order to identify boundaries between the eTeach client sessions, and ultimately be able to develop a session model, we first analyzed the distribution of OFF times. In simulated sessions, the inter-session OFF times usually has a very distinct distribution from the intra-session OFF times, which becomes clear when the overall distribution of OFF times is analyzed. However, for eTeach, no clear peak in the distribution of OFF times was observed in any of many periods analyzed. Thus, we found no indication of how to identify client sessions. The only insight obtained is that OFF times are heavily skewed, with typically 90% of OFF times less than 4 minutes and a few OFF times between 4 and 20 minutes.

For the analysis in this section, we define a session to be a sequence of requests from the same IP address (same client) to the same file such that each OFF time is not greater than 20 minutes. Similar results are obtained for a threshold of 4 minutes. Given this definition of a client session, this section provides the last component of our workload characterization, a session model, which includes the distribution of session inter-arrival times, the average number of interactive requests per session, the frequency of each interactive request type (e.g., pause, resume, jump forward, etc) and the distribution of ON and OFF times in the session. Unlike some previous characterizations of interactive media requests [78, 79, 109], these characteristics are analyzed for different days and for each file that had more than 20 sessions during the given day.

Table 11: Summary of Interactive Requests per Session per File in eTeach

File Size (min)	# Sessions	Avg # Interactive Requests / Session (min, typical, max)	% Pause (min, typical, max)	% Jump Forward (min, typical, max)	% Jump Back (min, typical, max)
0-5	992	0.35, 0.6-1.0, 1.8	5, 14-30, 55	0-3.5, 35	29, 45-95
5-15	449	1-3, 5.5	30, 40-55,70	12, 20-35, 50	11, 20-35
15-25	517	3-6	35-50, 60	15, 20-35, 45	20-35
30-35	411	2.5,4-5, 9	30-40 & 60-75	10-20 & 40-50	15-30

Recall that the eTeach interface available at the time of this study provides only pause, resume and jump into specific points in the video. Fast forwarding and rewinding are not directly provided by the interface, though they are available in the Windows Media Player interface. However, less than 0.5% of the interactive requests recorded in our logs are for fast-forwards and rewinds. Table 11 shows, for each other type of interactive request, the range of frequency typically observed over all sessions for a given file in a given file size range, as well as the minimum and maximum observed frequency if outside the typical range. As in [79], the average number of interactions per session increases somewhat with the file size. Although the average varies from day to day, it is always below 10 interactive requests per session, similar to the workload in [79], where 83% of the sessions have four or fewer interactions. For short videos, jump backward is the most common client interaction. As the file size increases, pauses become more common and, as the workload analyzed in [78], the fractions of jump forward and jump backward are approximately the same. In contrast, in the workload analyzed in [109], jump forward is the most common client interaction.

The distributions of session ON times depend on the file size. The best models are shown in Table 12. It is most commonly characterized as exponential with mean less than 1 minute for short videos (less than 5 minute long) and become heavier tailed as the duration of the video increases. OFF times have similar distributions, summarized in Table 13, with exponential (and sometimes

Table 12: Distribution of ON Times per File in eTeach

File Size (min)	Distribution		Mean (minutes), CV	
	Most Observed	**Other**	**Min Mean, CV**	**Max Mean, CV**
0-5	Exponential	Pareto	0.36, 0.4	0.87, 1.01
5-35	Pareto, Weibull	Lognormal, Gamma+Pareto	0.9, 1.03	3.7, 2.12

Table 13: Distribution of OFF Times per File in eTeach

File Size (min)	Most Observed Distributions	Mean (minutes), CV	
		Min Mean, CV	**Max Mean, CV**
0-5	Pareto, Exponential	0.46, 0.89	1.0, 1.83
5-35	Pareto, Lognormal, Weibull	0.43, 2.05	2.1, 2.76

Pareto) for small files and Pareto, lognormal or Weibull for larger files. Similar distributions for ON and OFF times were observed for the workloads in [109, 142].

In eTeach, the number of sessions for each file during any period of stationary client arrival rate is not sufficiently large to determine the distribution of inter-session times for each file, even for the most requested files on the high load days. Thus, curve fitting is used to analyze eTeach inter-session times for *all* sessions, during periods of stable session arrival rate. An example is shown in Figure 17. Unlike the Poisson session arrival process in BIBS and other workloads, the distribution of session arrivals at eTeach observed over different periods is heavier tailed with mean μ varying from 1.43 to 5.9 minutes and coefficient of variation CV varying from 1.16 to 1.48. We found that a good fit for the distribution of our data is either a Weibull distribution or a combination of Weibull for the body and Pareto for the tail. However, given that the coefficient of variation of the distributions is not much higher than 1, the assumption of Poisson arrivals used for the development of the CDN design techniques in this thesis may not be too inaccurate for eTeach sessions.

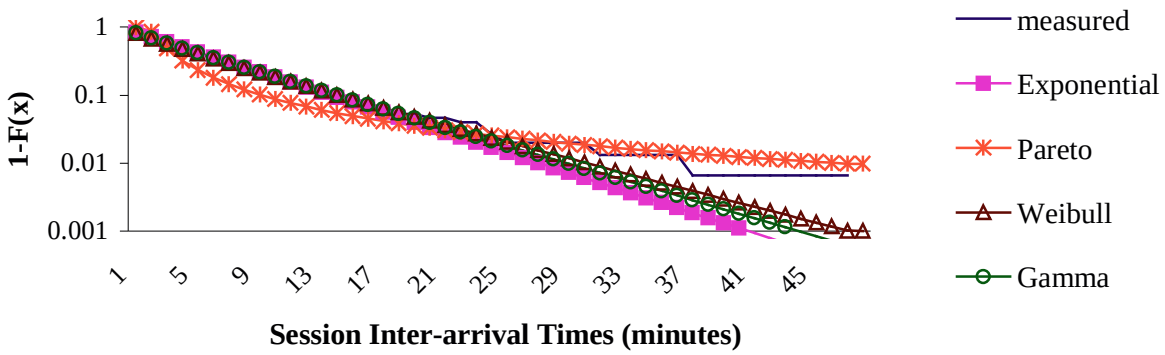


Figure 17: Example Distribution of Session Inter-Arrival Times in eTeach
(October 3rd, 9am – 11pm)

4.5 Summary of the Chapter

This chapter provided a thorough characterization of two educational streaming media servers with particular emphasis on meaningful statistics. New aspects of the workload characterization include: (a) analysis of four types of high load days, (b) request rate as a function of time, (c) distribution of overall and per-file request inter-arrival times during periods of stationary arrival rate, (d) changes in relative file popularity from day to day and from hour to hour, (e) distribution of file access frequency during each day, (f) distribution of file segment access frequency for files with different popularity rank, (g) fraction of accesses to cold files and cold file segments and (h) distribution for media delivered per request during periods of stationary average media delivered per request. We also analyzed session characteristics including arrival process, media ON and OFF times and fraction of each type of interactive request.

We found a high degree of similarity in the measures that could be analyzed for both servers. In particular, file access frequency per day (or during periods of approximate stable file popularity) at both servers can be modeled by the concatenation of two Zipf-like distributions, and the distribution of media delivered per session (or per request) depends on the file size and is heavier tailed for longer files. Furthermore, a large fraction of eTeach requests as well as BIBS sessions

have short duration. The only characteristic measured in both servers for which different models were found is the distribution of inter-arrival between sessions during periods of approximate stationary session arrival rate. The distribution that best fits the BIBS data is exponential, but the eTeach distribution is heavier tailed. We note, however, that new session requests are not explicitly identified in the eTeach log. Thus, session arrivals were identified heuristically for our analysis. The distribution of time between interactive requests within eTeach sessions is heavy tailed. Table 14 summarizes the most observed distributions for each characteristic analyzed in this chapter.

We found for both server workloads that, although arrival rates tend to be stable for long periods, relative file popularity changes significantly from day to day and from hour to hour, especially for BIBS. The implication of this fact for CDN design is that the system configuration (e.g. server content and routing) may need to be updated in response to those changes. The frequency of updates should be based on the cost of modifying the configuration to reflect the changes in the workload compared to the cost increase if the CDN configuration is not changed.

On both servers, a significant fraction (e.g., 50%) of new content (files or file segments) accessed each hour is not accessed again for four or more hours, which implies that caching content with no prior knowledge of its access rate may result in significant wasted disk i/o bandwidth and, consequently, in poor cache performance.

In eTeach, the frequency of accesses to each ten-second file segment is approximately uniform for the most frequently accessed files, and is skewed towards earlier segments for less frequently accessed files during each day. This is true for the files accessed on each day analyzed. The implications of these results are (a) measuring access frequency for only a few segments (instead of for every segment) may be enough to determine which segments to cache and (b) for unicast CDNs, full file caching of the most popular files might be the best strategy.

Table 14: Summary of Workload Characterization

Workload Characteristic	Most Observed Distribution
Inter-arrival Times (overall and per file, during periods of stationary arrival rate)	BIBS (sessions) Poisson eTeach (interactive requests) Pareto
File Lengths (% of requests to each file duration)	BIBS 13% to 0 – 5 mins 67% to 50 – 55 mins eTeach 27% to 0 – 5 mins 35% to 5 – 20 mins 35% to 20 – 35 mins 3% to 35 – 55 mins
File Access Frequency (per day) (BIBS and eTeach)	Concatenation of two Zipf-like distributions
File Segment Access Frequency (eTeach)	Most Frequently Accessed Files Uniform Least Frequently Accessed Files Skewed towards earlier blocks
Media ON Times within Session (eTeach)	File Length < 5 min Exponential Larger File Lengths Weibull or Pareto
Media OFF Times within Session (eTeach)	File Length < 5 min Exponential or Pareto Larger File Lengths Weibull / Pareto / Lognormal
Media Delivered Per Session (BIBS)	File Length < 5 min Lognormal Larger File Lengths Gamma+Pareto
Average Number of Interactive Requests Per Session (eTeach)	File Length < 5 min 0.6 – 1 Larger File lengths 3 – 6
Most Common Client Interactions (eTeach)	File Length < 5 min jump backwards (45 – 95%) pause (14 – 30%) Larger File Lengths pause (40 – 75%) % jump backwards ≈ % jump forwards

Our analysis also provides example profiles of client interactive behavior, which can be used for creating more realistic client interactive access models.

Finally, we found that despite the large number of interactive requests to very short segments (average stream duration of 1.5–2 minutes), the bandwidth savings from multicast delivery of the three most popular files on several days in eTeach can be up to 30%.

Chapter 5

Caching Policies for Unicast Streaming CDNs

In the context of unicast delivery, server placement, server selection and routing for both web and streaming CDNs have been fairly explored by other parallel work [13, 14, 15, 36, 71, 75, 85, 93, 96, 110, 115, 116]. Open issues in caching for unicast streaming are explored in this chapter. One main goal of this thesis is to determine whether the Resource Based Caching (RBC) algorithm [137, 138, 139], the best previously known caching policy, outperforms a much simpler rate driven caching policy (that will be called Currently Most Popular, or CMP), which stores in the cache only the files that have the highest measured access frequencies and thus minimizes the total delivery cost, with the possible exception of adding interval caching [51], which may further improve the performance. We also propose and evaluate two other new caching policies, Pooled RBC and CMP/IC, based on improvements to the basic RBC and CMP policies. By comparing the performance of these policies, we evaluate whether the delivery cost of streaming CDNs can be significantly reduced if the proxy servers store, in addition to full objects, intervals of data for the currently active requests.

Previously reported results seemed to indicate that RBCis a very promising caching policy. It combines full file caching with interval caching and was shown to outperform the simpler Least Frequently Used (LFU) policy over all experiments considered in [138]. Rejaie *et al* even

suggested future extensions of RBC for caching of multi-layered objects [122]. However, the design space explored by the experiments in [138] is very narrow, and we have verified that some of the reported results are incongruous, as discussed later in the chapter. These observations motivate a reevaluation of the performance of RBC

The remainder of this chapter is organized as follows. Section 5.1 describes the interval caching and RBC policies. Section 5.2 describes the newly proposed caching strategies as well as improvements on the basic RBC. Section 5.3 reviews the evaluation methodology. The main performance results as well as a discussion on the effectiveness of interval caching are provided in sections 5.4 and 5.5, respectively. Section 5.6 summarizes the chapter.

5.1 Description of Interval Caching and Resource Based Caching

This section describes Interval Caching (section 5.1.1) and Resource Based Caching (RBC) (section 5.1.2), two previously proposed caching strategies that are further evaluated in this chapter.

5.1.1 Interval Caching (IC)

Interval caching [51] was proposed in order to satisfy multiple client requests that arrive close in time, by caching in main memory *intervals* of streaming media file data. Intervals exploit temporal locality and the sequential access pattern of streaming objects to use the limited cache space to store data that is guaranteed to be needed soon in the future at the possible expense of high cache read and write bandwidth requirements. It works as follows: if two client requests for file f arrive close together in time, the data delivered to the first client may be retained in memory until it is delivered to the second client. After this time, the buffer used to hold the data in memory is released and reassigned to a future block. The two requests form an interval, with the first request

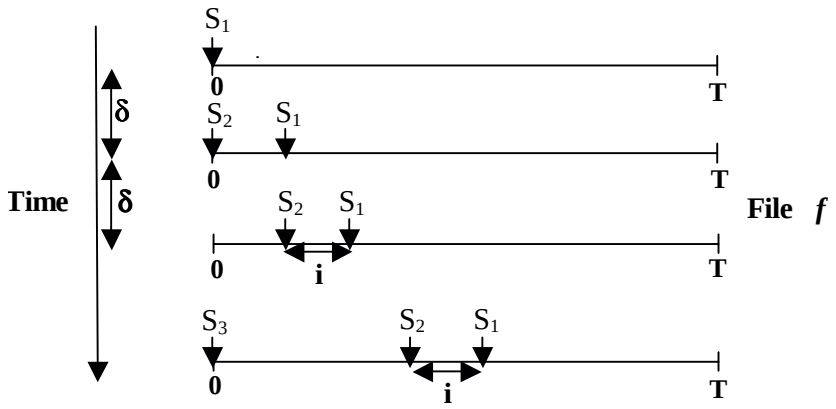


Figure 18: Interval Caching

called the *writer* and the second request called the *reader*. Figure 18 illustrates this relationship: the interval *i* consists of the pair [S_1, S_2]. S_1 is the writer stream, which retrieves data blocks from the origin server and stores it in the memory cache, at the proxy server. S_2 is the reader stream, which reads the blocks from the cache and releases the buffer. The interval is created when S_2 arrives, δ minutes after S_1. At this time, S_1 starts caching the data it receives from the origin server. S_2 needs to retrieve the initial δ minutes of playback from the origin. At time δ minutes after its arrival, S_2 starts retrieving blocks from the cache. As data blocks are read by S_2, the cache space used to hold them is released and can be reused by S_1. The caching strategy is to cache the set of the smallest intervals that can fit in the main memory cache, since they contain the data that will be accessed soonest in the future.

Interval caching was first proposed in [51], generalized to handle short and long file workloads in [53] and further analyzed in [50, 54]. Parallel work proposes similar memory caching policies in the context of multimedia database systems [90, 107].

5.1.2 Resource Based Caching (RBC)

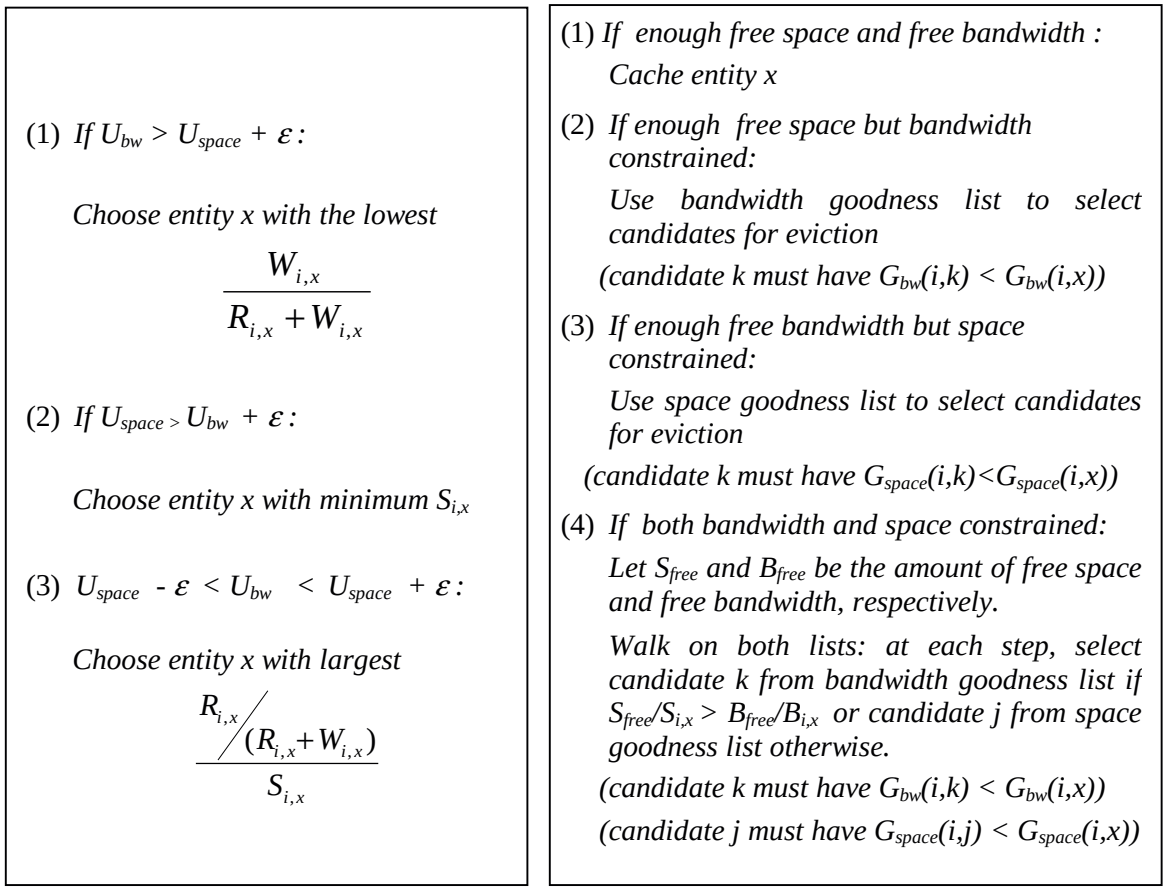
The Resource Based Caching (RBC) algorithm is a disk-based policy that characterizes each file object by its space and bandwidth requirements and a measure called *caching gain*. RBC caches a

Table 15: Parameters Used in RBC

Notation	Description
$R_{i,file}$	Estimated average number of clients that request file *i* during its delivery time
$R_{i,x}$	Current number of clients receiving but not writing entity *x* of file *i* ($x \in$ {interval, run}).
$W_{i,x}$	Current number of clients receiving and writing entity *x* of file *i* to disk ($x \in$ {interval, run , file})
r_i	Bit rate for receiving file *i*.
$B_{i,x} = (R_{i,x}+W_{i,x}) \times r_i$	Bandwidth requirement of entity *x* of file *i* ($x \in$ {interval, run , file})
$S_{i,x}$	Size, in bytes, of entity *x* of file *i* ($x \in$ {interval, run , file})
$g_{i,x} = R_{ix} \times r_i$	*Caching gain* for entity *x* of file *i* ($x \in$ {interval, run , file})
$G_{space}(i,x) = g_{i,x}/S_{i,x}$	*Space goodness* measure for entity *x* of file *i*
$G_{bw}(i,x) = g_{i,x}/B_{i,x}$	*Bandwidth goodness* measure for entity *x* of file *i*
U_{space}	Current disk space utilization
U_{bw}	Current disk bandwidth utilization

mixture of full files and intervals and tries to efficiently utilize the limited resource, either disk space or disk bandwidth. The development of the RBC policy was guided by two key goals: (1) keep disk space utilization and disk bandwidth utilization approximately equal, and (2) replace the entity in the cache that is estimated to be needed again farthest in the future. The value of the first goal is somewhat unclear; a more intuitive goal is simply to maximize the disk bandwidth that is used to deliver content to the clients. The second goal leads to optimal caching policy performance if the estimate of which data is needed farthest in the future is accurate *and the system is not bandwidth constrained*.

RBC uses the parameters defined in Table 15. When a client request for file *i* arrives and file *i* is not fully cached, the RBC policy follows two steps. In Step 1, the system selects the granularity (interval, run or full file) of file *i* that should be considered for placement in the cache, using the criteria in Figure 19(a). Runs are extensions of intervals, where a set of adjacent intervals are grouped together, and are particularly interesting when both bandwidth and space are limited



a) Step 1: Select Granularity of Entity *x* **b) Step 2: Caching Decision for Entity *x***

Figure 19: Resource Based Caching Algorithm: Handling a Request for File *i*.

because they amortize the write overhead over multiple readers and use less space than the entire file. In Step 2, described in Figure 19(b), the system *evaluates* whether the selected entity should be cached. The entity (i,x) selected in Step 1 is cached only if space equal to $S_{i,x}$ *and* bandwidth equal to $B_{i,x}$ can be allocated to it, as dictated by Step 2 in Figure 19(b).

RBC only caches a full file if there is enough unallocated bandwidth (or enough bandwidth is made available after removing entities from the cache) to be reserved for the estimated *average* number of clients that simultaneously request the file. Furthermore, it does not allow unused bandwidth that is currently allocated to one file to be used to serve a request to another file. This

might be a drawback if bandwidth is scarce, especially given that statistical fluctuations in the actual number of concurrent requests imply that allocated bandwidth is often not fully utilized. RBC also characterizes each entity by two measures called *space goodness* and *bandwidth goodness,* defined in Table 15. These measures are used to keep the cached entities in two separate sorted lists. The need to maintain multiple sorted lists with multiple types of entities for the RBC policy results in a complex implementation. Tewari *et al* [138] compare RBC against the much simpler LFU policy [130] and conclude that RBC provides a better *byte hit ratio* in the cache. However, the region of the system design space considered is narrow (in particular, they consider systems with cache sizes up to only 16GB). Furthermore, some of their results are inconsistent with simple formulas for bandwidth and space needed to deliver the specified workload. By comparing the average duration of the files that fit in the cache, request arrival rate and the average number of simultaneous streams delivered by the proxy (calculated from the proxy byte hit ratio), one could determine that it violates Little's Result [94]. Moreover, due to the significant difference in complexity, the relative performance of LFU and RBC should be evaluated in the context of their relative complexity.

Finally, the previous literature [137, 138] omits some key details of the RBC policy. Fortunately, one of the policy inventors has given us the missing information needed to perform our policy comparisons. Some of these details are discussed in section 5.2.1, together with newly proposed improvements on the RBC policy.

5.2 New Caching Strategies

This section introduces several new caching strategies for unicast streaming. Section 5.2.1 describes some changes proposed for the complex RBC policy in an attempt to make it more intuitive and possibly more efficient. Further improvements lead to the development of the Pooled

RBC policy, described in section 5.2.2. Section 5.2.3 describes the Currently Most Popular (CMP) caching algorithm, a rate driven frequency based policy, and relates it to the traditional LFU caching policy. Finally, a hybrid caching policy that combines CMP with simple interval caching is introduced in Section 5.2.4.

5.2.1 Improvements to the RBC Policy

This section proposes two changes to RBC: a replacement for the original rule for selecting the granularity of the entity to be cached, i.e., Step 1, described in Figure 19(a), and a rule to handle the case when a full file is selected to be cached but there are already intervals/runs of the file in the cache. The three criteria used in Step 1 of the RBC algorithm are designed to keep bandwidth and space utilization approximately equal when choosing the granularity of the entity to cache. Given this goal, it is not clear why RBC selects the entity with minimum $S_{i,x}$ when U_{space} is greater than $U_{bw} + \varepsilon$, rather than the entity with maximum $R_{i,x}/S_{i,x}$. Furthermore, the sets of rules used in Steps 1 and 2 are different and not always consistent. One could imagine the situation where U_{space} is greater than $U_{bw} + \varepsilon$ which would result in the selection of the entity with minimum $S_{i,x}$ in Step 1, but the system could be bandwidth constraint with respect to that entity (fall under option (2) in Step 2). We have experimented with a conceptually simpler and perhaps more intuitive rule for Step 1, namely always select the entity with the largest value of $R_{i,x}/S_{i,x}$, regardless of the values of U_{bw} and U_{space}. This rule selects the entity that maximizes the expected amount of bandwidth used to deliver data to clients per unit of cache space. In all cases, we have determined that (perhaps surprisingly) this change does not affect the performance observed for RBC.

Whenever a full file is selected for caching and there are intervals/runs of the file already in the cache, the policy in [138] computes the additional space and additional bandwidth needed to

cache the full file. If they can be pre-allocated, the full file is cached. The intervals are still kept in the cache while the full file is being cached, but if the full file is evicted from the cache, the intervals are also evicted. In other words, the intervals are completely assimilated by the full file for the purpose of future replacement decisions. For the results reported in this chapter, the same algorithm for deciding whether to cache the full file is used. However, if the full file is later evicted, but the intervals are still active, the intervals are retained in the cache. This is a slightly more complex and general rule but, under certain scenarios, it may be important to keep the assimilated intervals/runs in the cache if their original goodness values dictate so. We experimented with evicting the intervals when the file is evicted, and, surprisingly, found no noticeable impact on policy performance for the experiments described in this chapter.

The RBC policy analyzed in this work has the more general rule to handle the assimilation of cached entities by entire files. The modified Step 1 is used only in the Pooled RBC policy, described next.

5.2.2 Pooled RBC

The strict and independent scheme of pre-allocation of disk bandwidth used in RBC might actually hurt the policy performance if bandwidth is a limited resource, as mentioned in section 5.1.2. Based on this insight, the new **Pooled RBC** policy is proposed. In this policy, full file entities share their bandwidth allocation. Whenever a new full file is added to the cache, its bandwidth allocation is added to the *bandwidth pool*. When a request for a cached file arrives, a new stream is obtained from the bandwidth pool and assigned to that request. When it finishes delivering the file, the bandwidth is returned to the pool and can be assigned to a new request, even if that request is for a different cached file. If a request for a cached file arrives and all of the pooled bandwidth is in use, Pooled RBC operates just like RBC. At any given point in time, Pooled RBC allows a file with less

than its average number of clients to donate bandwidth for delivering another file that has more than its average number of clients. Pooled RBC uses the simplified Step 1 in addition to the new assimilation rule described in the previous section. Note that Pooled RBC and RBC have the same complexity in both implementation and run-time.

5.2.3 Currently Most Popular (CMP)

A much simpler caching policy, compared to RBC and Pooled RBC, is the Currently Most Popular, or CMP policy. CMP is a rate driven frequency based caching policy that stores in the cache only the objects with the highest measured request rates. Note that, conceptually, LFU and CMP might be considered similar. However, CMP and LFU *may* differ with respect to a key implementation issue, namely, the handling of cache misses. CMP only stores an object in the cache after enough information on its access rate has been gathered to determine that it is cost-effective to cache it, or, in other words, it is one of the currently most frequently accessed objects. In the context of memory caches [130], for which it was originally proposed, LFU, unlike CMP, always caches an object upon a miss and then starts gathering information on its popularity. It is not clear, from the literature, whether the handling of cache misses is the same for different workloads such as web and streaming media. In particular, it is not clear whether LFU is implemented differently from CMP in [138].

We also point out that, regarding the method of estimating object access rate, LFU is *usually* associated with the use of reference counts [130]. CMP, on the other hand, may use any estimate of file access rate used in previously proposed caching policies [3, 86,106, 130, 122, 123, 148]. Determining the best estimate is a complex problem, which involves several tradeoffs (e.g., accuracy during periods of stable access frequency versus fast detection to changes), and remains an open issue for future work.

5.2.4 Hybrid Currently Most Popular / Interval Caching (CMP/IC)

Since CMP is a much simpler policy to implement compared to RBC and Pooled RBC, two hybrid strategies are proposed with the goal of possibly improving its performance by storing simple intervals in a small, separate cache. The first strategy uses main memory to cache intervals. The second strategy reserves a small fraction of the available disk space for interval caching. In both approaches, only simple intervals are cached; runs are not considered as possible caching entities. By comparing CMP and CMP/IC, we will be able to assess the effectiveness of interval caching for the experiments performed.

The benefit of interval caching in main memory is assessed by evaluating the performance of the disk-based Pooled RBC and CMP on a system that is augmented with a main memory cache managed by the simple interval caching policy [51]. Two systems are considered, each one with a disk and a memory cache of size *M*. The primary policy, either CMP or Pooled RBC, is used to manage the disk cache and interval caching is used in the memory cache. Those systems are named CMP/IC_{mem} and Pooled RBC/IC_{mem}, respectively. The interaction between both caches depends on the primary policy. In both systems, whenever a request to a cached file arrives and there is enough disk bandwidth, it is served from disk. Otherwise, we try to cache an interval in memory so that the memory cache can be used by the most popular files without losing the advantages of also caching them entirely in disk. If the requested file is not in the cache, then we first try to create an interval in memory. If not possible, then the primary policy is used: Pooled RBC tries to create an entity in the disk cache; CMP decides, based on the current estimate of popularity of the requested file, whether to cache it. When an interval is removed from the memory cache, Pooled RBC treats the interval as a disk entity and tries to cache it on disk.

A different strategy is to use a small *fixed* fraction of the disk space for interval caching instead of main memory. The goal is to determine whether CMP can be improved if some fraction *s* of disk space (e.g. 2%-5%) is reserved to cache intervals. In particular, one key issue is how large *s* must be so that this hybrid strategy, CMP/IC_{disk}, performs at least as well as Pooled RBC. Note that this hybrid disk caching policy tries to cache only the smallest intervals for the files that are *not* in the CMP cache.

5.3 Evaluation Methodology

The main assumptions and parameters for evaluating CDN designs and, in particular, the performance of caching policies, were discussed in chapter 3. We point out that the relative performance of different unicast caching policies does not depend on the number of proxies and thus can be evaluated by considering only one proxy and one origin server and varying the system and workload parameters widely in order to cover a large variety of systems. Table 16 gives the set of system and workload parameters that will be considered in the following sections to evaluate the relative performance of RBC, Pooled RBC, CMP and CMP/IC. Note that it contains the same derived parameters shown in Table 3 (chapter 3), except for β, the ratio of the average cost of a proxy stream to the average cost of an origin stream, which does not impact the optimal proxy content for unicast CDNs.

Furthermore, since only one proxy needs to be considered, we use a single parameter N to represent the total client request rate and a single parameter θ for the overall distribution of file access frequencies. We also point out that changing the number of files n has a similar effect on relative file popularity as changing the value of the parameter θ. Thus, only one parameter needs to be varied to evaluate the performance of different strategies. We will hold θ constant and vary n.

Table 16: System and Workload Parameters for Evaluating Caching Strategies for Unicast

Parameter	Description
N	Total client request rate, in units of requests per average file playback time ($N=\lambda T$)
n	Number of files
θ	Parameter for the Zipf-like distribution used to model the file access probabilities
C	Proxy cache size, expressed as a fraction of the sum of all *n* media file sizes
B	Proxy disk bandwidth, as a fraction of the bandwidth needed to deliver the most popular files that can fit in the proxy storage space

We set θ = 0, as the high skew may favor interval caching and RBC. However, additional experiments show that the relative performance is the same for θ = 0.271, the parameter value observed for the frequency of movie rentals in video stores in [52].

Recall that in chapter 3 we proposed two methods for normalizing the total proxy disk bandwidth. In this chapter, we express the total proxy disk bandwidth, ***B***, as a fraction of the average bandwidth needed to deliver the most popular files that can fit in the proxy cache space, i.e., as a fraction of $r \times BW^*$, where *r* is the average file streaming rate and BW^* is the average number of active streams needed to deliver the data that is cached by the CMP policy. Given $\lfloor n \times C \rfloor$, the index of the least popular file that fits in the CMP cache, BW^* is computed as follows:

$$BW^* = N \times \sum_{i=1}^{\lfloor n \times C \rfloor} p_i \,, \tag{4}$$

where p_i is the access probability for file *i*, defined by the Zipf-like distribution. The number of concurrent streams at a given rate *r* that a disk can sustain was given in Table 2 (chapter 3) for two generations of disks. The number of concurrent streams per disk times the number of disks divided by BW^* is the value of *B*. Alternatively, since the load is assumed to be balanced across the disks, the ratio of the number of concurrent streams per disk to the number of concurrent streams needed

to deliver all of the content on the disk is also equal to *B*. For example, if the Seagate disk caches three two hour MPEG-1 (1.5 Mb/s) movies, and the total request rate for the three movies is ten requests per hour, twenty streams are needed to deliver the data cached on the disk and $B = 25/20 = 1.25$ (Table 2 shows that a Seagate disk can sustain 25 MPEG-1 streams). Similarly, if the Ultrastar disk caches eighty-seven thirty minute MPEG-2 (4 Mb/s) videos and the total request rate for the shows is equal to 120 requests per hour, 60 concurrent streams are needed to deliver the cached content and $B = 42/60 = 0.7$. Depending on the duration and streaming rate and the total request rate for the files stored on the disk, the value of *B* can be less than, equal to or greater than 1.

If $B = 1.0$, a proxy that implements the CMP policy is expected to have enough bandwidth to be able to serve most requests for the cached files, and will have bandwidth utilization close to 100%. Thus, such systems where $B = 1.0$ will be called *bandwidth balanced* systems. Similarly, in systems where $B < 1.0$, a significant number of client requests that hit in the CMP proxy cache may not be delivered from the cache due to limited disk bandwidth. Such systems will be called *bandwidth deficient* systems. When $B > 1.0$, in the CMP caching system, bandwidth utilization can be expected to be approximately equal to $1/B$, and thus these systems will be called *bandwidth abundant* systems. For the experiments in this paper, values of *B* equal to 0.75, 1.0, and 2.0 are considered as representative of these three types of systems.

For evaluating the relative performance of the caching policies proposed in this chapter, we will simulate the policies using a synthetic workload defined by the assumptions listed in chapter 3. In particular, we will further assume that the arrival rate and file access frequencies (given by the Zipf-like distribution) are fixed during the simulation and we will vary the parameters widely in order to cover as many different client workloads as possible. We point out that, since all the policies considered in this chapter rely on an estimate of access frequency, we expect the relative

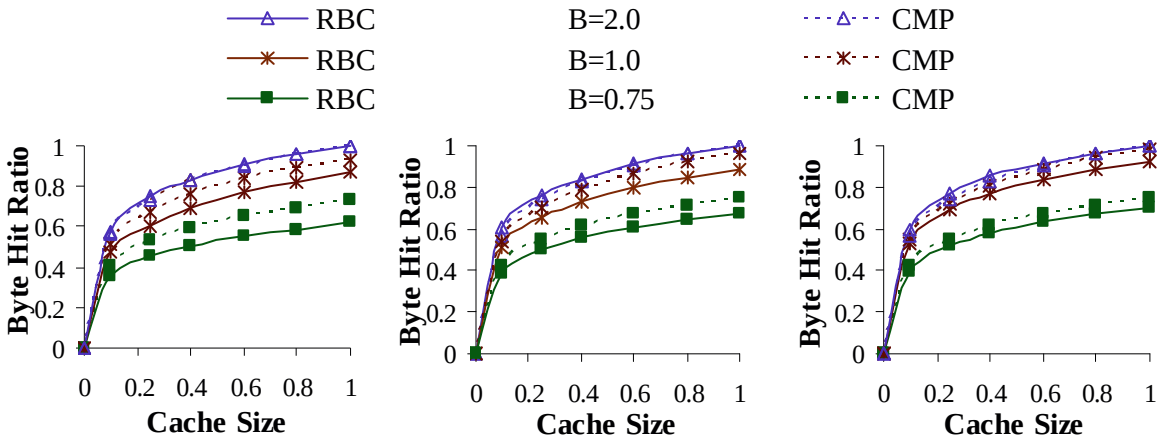
performance of them to remain (approximately) the same in the presence of dynamically changing file access frequencies (as long as they use the same estimate). In particular, the same relative performance of the policies, reported in the next section, was observed with extra experiments that assumed unknown access frequencies estimated with a weighted average of previous request inter-arrival times, as proposed in [138].

5.4 Performance Comparisons

This section evaluates the performance of the caching policies. Section 5.4.1 provides the main results on the relative performance of RBC, Pooled RBC and CMP. Unlike the results reported in [138], our experiments show that RBC only outperforms CMP, by a very small amount, if the system is bandwidth abundant. Section 5.4.2 evaluates CMP, Pooled RBC and CMP/IC. Our results show that the benefit of interval caching is very most and the simple CMP caching policy is the most cost-effective policy. The 95% confidence intervals for all results shown in this section are within ± 2% of the reported values.

5.4.1 CMP, RBC and Pooled RBC

The byte hit ratios of CMP and RBC for different values of C, B and N are shown in Figures 20(a)-(c). Two key conclusions are drawn from these results. First, the relative performance of RBC and CMP is very insensitive to the arrival rate. Though the performance of RBC improves as the arrival rate increases, consistently with the results reported in [138], the same relative ordering of the policies is observed for the various values of N. We also point out that the relative performance of the policies is also very insensitive to the number of files [9]. Thus, it appears that the relative performance of the policies is determined by C and B. Second, CMP outperforms RBC if disk bandwidth is a limited resource ($B \leq 1$); otherwise, RBC only *slightly* outperforms CMP. Given the

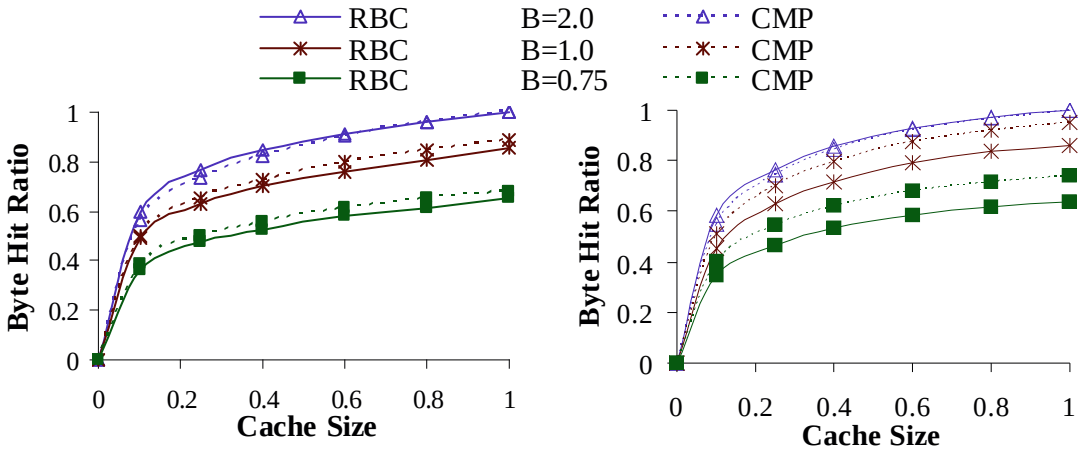


a) *N*=100 **b) *N*=450** **c) *N*=1530**

Figure 20: Performance Comparison of RBC and CMP (n = 100)

significant difference in complexity, CMP is more cost-effective than RBC. We point out that these results *qualitatively* agree with the results in [138], where the authors conclude that RBC outperforms LFU but compare the two policies only over a region of the design space for which $B > 1$. However, we find that RBC outperforms CMP by a much smaller amount in that region of the design space.

As discussed in chapter 3, a burstier arrival process, compared to Poisson (default assumption), may favor interval caching, and consequently RBC, because the number of smaller intervals eligible for caching is larger. We compared the performance of RBC and CMP assuming Pareto request arrivals with parameters $\alpha = 1.5$ and $k = 1$ [20]. Figure 21(a) shows that, although the burstier arrival process does improve RBC performance (the gap between the policies decreases), the relative performance between the two policies remains the same. We also evaluated the sensitivity of our conclusions to the assumptions of uniform file duration and uniform file streaming rate. We ran experiments with a heterogeneous workload in which files are partitioned into three different classes defined by the file length. A separate Zipf distribution is used to model



a) Pareto arrivals (α=1.5, k = 1)

b) Variable File Durations and Streaming Rates (N=330, n=300)

Figure 21: Further Performance Comparison of RBC and CMP

accesses to files of each class. Each class has 100 files and each file has an equal probability of having a streaming rate equal to either 1.5 or 4.5 Mbit/s. Class 1 contains 5-minute clips, class 2 contains 30-minute videos and class 3 100-minute videos. The access frequencies for each of the three classes are: 0.6, 0.3 and 0.1, respectively. Figure 21(b) shows that the relative performance of RBC and CMP is the same as for uniform file durations and streaming rates.

In order to better understand the relative performance of RBC compared to CMP, the average fraction of each file cached by RBC is measured. Figure 22 shows the fractions for C = 0.25, N = 450 and n = 100. When $B \leq 1$ (Figures 22(a) and 22(b)), like CMP, RBC caches mainly full files. However, RBC caches fewer files because, as discussed in section 5.1.2, unused bandwidth that is currently allocated to a file cannot be used for serving a request to another file. Statistical fluctuations in the actual number of concurrent requests for each file imply that allocated bandwidth is often not fully utilized, as discussed below. If bandwidth is abundant (Figure 22(c)), RBC slightly outperforms CMP by choosing to cache intervals and runs instead of some of the least popular files [138]. However, it is important to note that RBC still fully caches the most

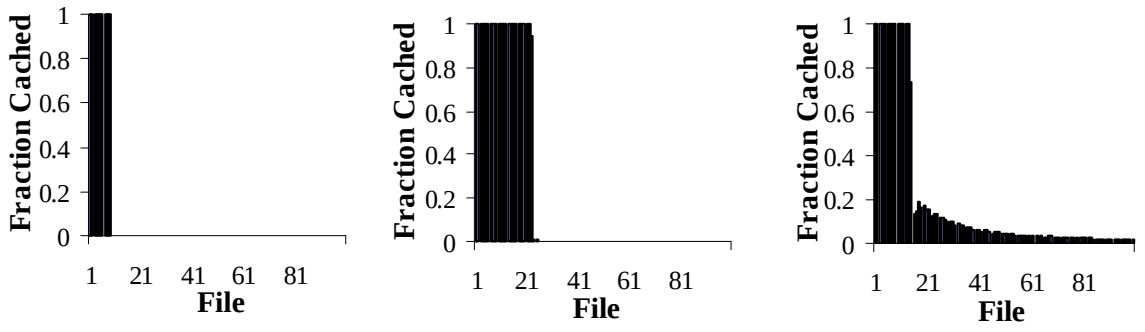


Figure 22: Average Fraction of Each File Cached by RBC (N = 450, n = 100, C = 0.25)

popular files and only caches intervals/runs of the less popular files. This is why, for $B > 1$, the difference in performance is small and disappears for larger cache sizes. As C increases, more files are fully cached by RBC and interval caching becomes less effective because there are not many small intervals for less popular files. Based on additional experiments that vary B from 1 to 2 and keep $C = 0.1$, we found that the byte hit ratio of RBC is higher than the byte hit ratio of CMP only if $B \geq 1.2$ and no further improvement on RBC performance is achieved if B increases beyond 1.4, because there are no sufficient cacheable intervals to use the extra bandwidth.

Another key result is that, although RBC tries to equally utilize disk bandwidth and space, the two utilizations are quite different when $B \neq 1$, as shown in Figure 23. In particular, for $B \leq 1$, although RBC allocates all the available bandwidth to the cached files, it uses only about 90% of it, due to the frequent discrepancy between the number of active requests per file and the amount of bandwidth allocated to each file. On the other hand, CMP uses nearly 100% of the disk bandwidth to deliver data from the cache.

Pooled RBC is proposed as an improvement over the strict and independent scheme of pre-allocation of disk bandwidth used in RBC. Recall that in Pooled RBC, full files share their bandwidth allocation. Figure 24 shows that Pooled RBC performs much better than RBC and has a

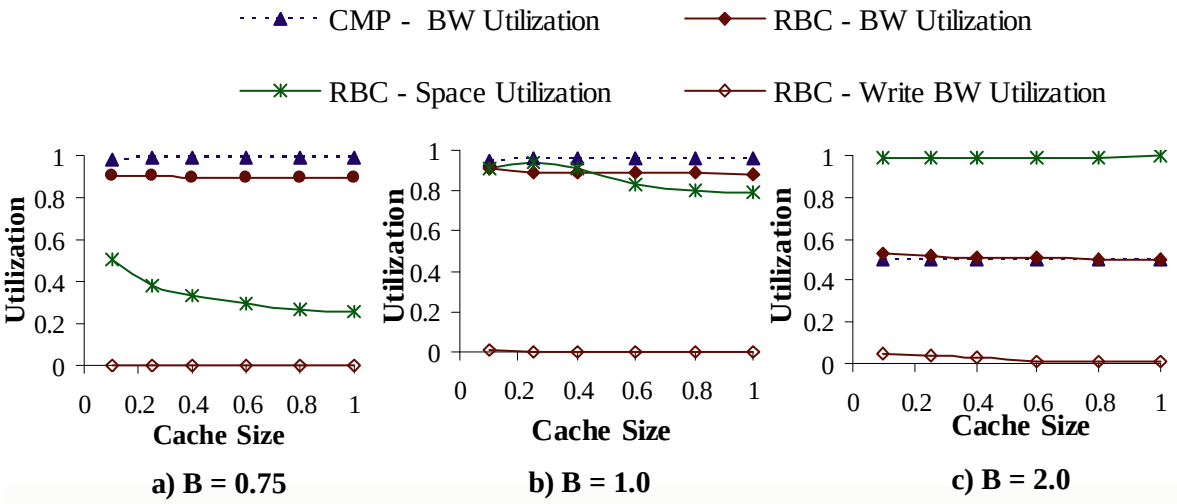


Figure 23: Space and Bandwidth Utilization for RBC and CMP (*N* =450, *n* = 100)

byte hit ratio much closer to CMP for $B \leq 1$. For $B = 2.0$, Pooled RBC performs the same as RBC because the partitioned bandwidth allocation does not hurt the performance of RBC when there is excess of disk bandwidth.

5.4.2 Hybrid CMP/IC

This section investigates the performance of the two hybrid CMP/IC strategies, namely, CMP/IC_{mem} and CMP/IC_{disk}, designed to improve CMP performance by storing intervals in a small, separate cache, either in main memory or in a fraction of the disk space. Figure 25 compares the performance of CMP/IC_{mem} and Pooled RBC/IC_{mem}, as defined in section 5.2.4, for memory cache size $M = 0.25\%$ of the total file data. The overall key conclusions are: (a) interval caching in main memory improves, somewhat, the performance of both policies and (b) the CMP/IC_{mem} policy performs practically as well as or better than Pooled RBC/IC_{mem} for all values of *B* and *C*.

Figure 26 shows the performance of CMP, Pooled RBC and CMP/IC_{disk}, if the fraction *s* of disk space reserved for caching intervals is equal 5%. For all values of *B* and C, our results show

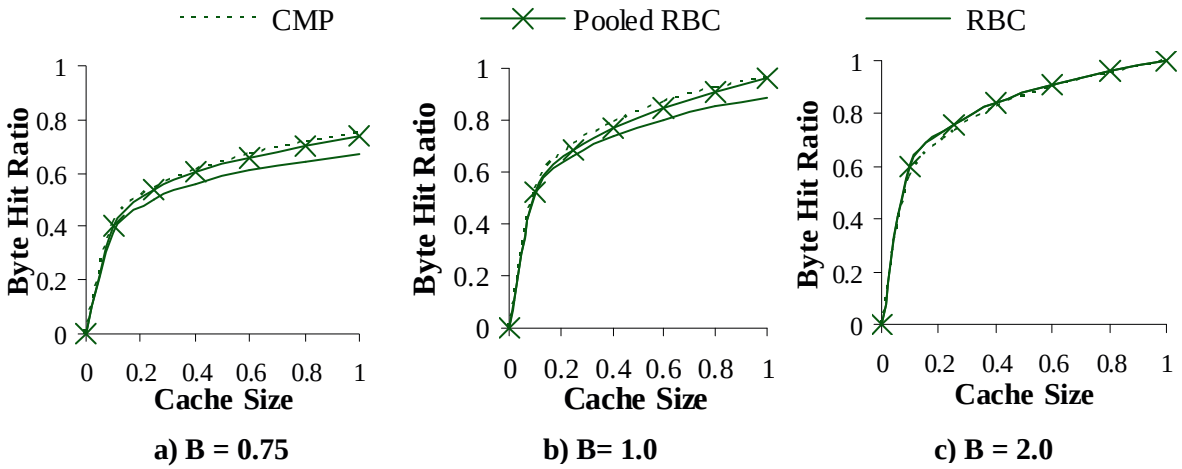


Figure 24: Performance of Pooled RBC Compared to RBC and CMP (N = 450, n = 100)

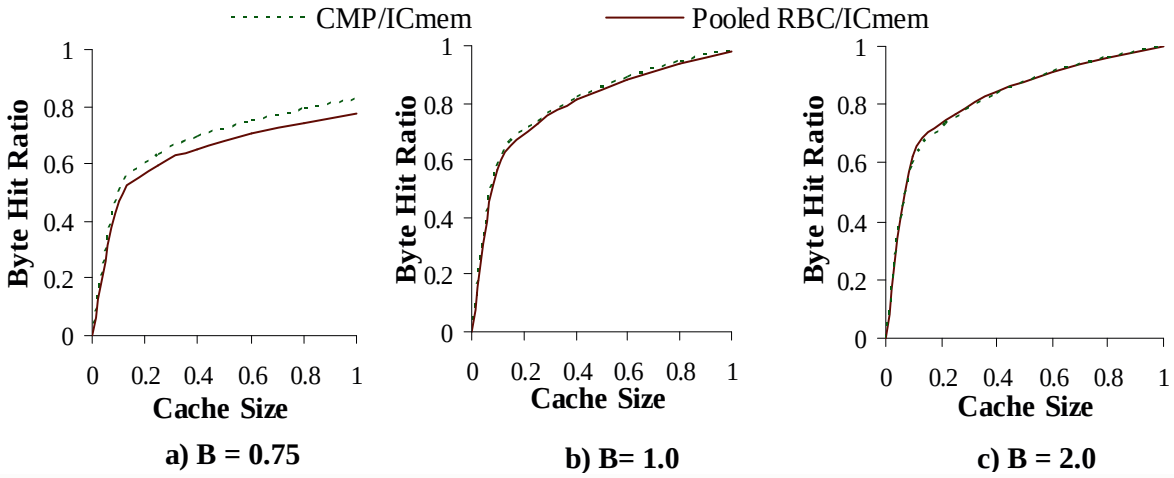


Figure 25: Performance of Hybrid Pooled RBC/IC$_{mem}$ and CMP/IC$_{mem}$

(N=450, n=100, M=0.25%)

that: a) CMP/IC$_{disk}$ performs as well or better than Pooled RBC and b) the byte hit ratios achieved with CMP and with CMP/IC$_{disk}$ are indistinguishable. Furthermore, reserving a larger fraction of disk space for interval caching does not result in significant improvement. In fact, the performance may actually degrade. For example, s = 10% hurts performance for cache size $C \geq 0.8$.

Concluding, the hybrid CMP/IC policies perform as well or better than Pooled RBC for all parameter values, with the advantage of being much simpler to implement and execute. Moreover,

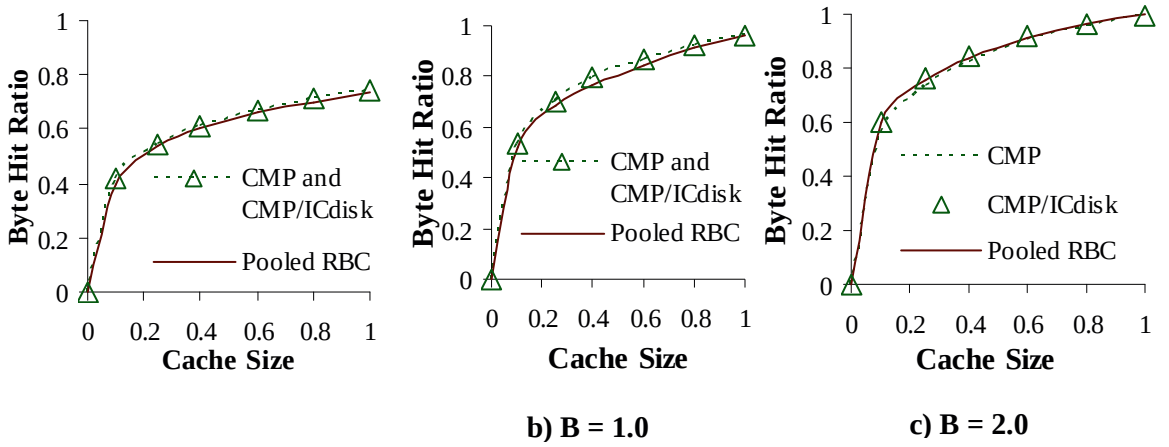


Figure 26: Performance of the Hybrid CMP/IC$_{disk}$ (N = 450, n = 100, s = 5%)

compared to the even simpler CMP, the improvements from interval caching are very modest. Thus, the CMP policy is a good compromise between simplicity and efficiency.

5.5 Interval Caching for Streaming Media

The results described in the previous section lead to the key observation that the benefit from interval caching is very modest. Most of the available disk cache space and disk bandwidth is always better utilized if used to fully cache the most popular files, leaving the remaining resources (if any) for caching intervals over less popular files. This was observed over a wide range of parameter values and for both Poisson and the burstier Pareto arrival processes. Furthermore, even if performed in main memory, the benefit from interval caching is modest considering that main memory is a limited resource that, in practice, may be better utilized as a buffer cache for the disk accesses as well as to perform other streaming related functions (e.g., smoothing).

One possible advantage of interval caching not explored in this work is the robustness in the presence of varying file access frequencies. If a file becomes suddenly very popular, memory-based interval caching can be used to serve some requests from the proxy while enough information becomes available to decide whether the full file should be cached in the CMP disk

cache. However, also in this scenario, we expect the performance improvement to be modest as only a few requests would benefit from interval caching, assuming that an efficient method for detecting changes in file access frequencies is used.

Furthermore, is a system with an interactive client community, where clients may request different segments of the file, interval caching may not help at all if the blocks retrieved for different requests that arrive close in time do not overlap. For instance, in chapter 4, we show that a large number of consecutive requests to eTeach files are for very small and non-overlapping segments. For this workload, interval caching would not help. Instead, a simple full file CMP caching may be much more effectively, especially given the uniform segment access frequencies for the most popular files, as discussed in chapter 4.

We also do not expect interval caching to be effective for multicast delivery. Take, for instance, the efficient Bandwidth Skimming multicast delivery protocol [60, 61, 62]. Each client receives data by listening to possibly two separate streams, one created when its request arrived at the server and the other, the closest target stream, created earlier, in response to a previous request. The two streams merge as soon as their clients reach a common point at the playback. If the same client listens to both streams, an interval between them would not make sense since the client receives data from both of them. Thus, it seems likely that the eligible intervals for caching would most frequently be established between streams that are far apart and do not have clients in common. Those intervals are long and, thus, not interesting for caching.

5.6 Summary of the Chapter

This chapter evaluated the performance of caching algorithms for unicast streams and determined that, due to implementation simplicity as well as delivery cost, the new and much simpler Currently

Most Popular (CMP) caching policy, which stores at the proxy only files with highest measured access rates, outperforms the best previously proposed policy, called RBC. These results are based on a more complete analysis, which explores a larger system design space than previously considered. Based on the insights obtained with this analysis, two new caching policies, Pooled RBC and CMP/IC, were proposed in an attempt to improve RBC and CMP performance, respectively. Pooled RBC includes a key improvement to the original RBC policy in which bandwidth pre-allocations are shared among all cached files. The simpler hybrid CMP/IC combines CMP with interval caching, performed either in main memory or on a separate small portion of the disk cache. The main results showed that the hybrid CMP/IC_{disk} has performance at least as good as Pooled RBC, as well as the advantage of being much simpler to implement. However, the even simpler CMP policy delivers essentially the same performance as Pooled RBC and CMP/IC_{disk}. Overall, the benefit of interval caching is at best very modest. Since interval caching is expected to be even less beneficial for CDNs that use scalable delivery protocols, it is not further explored in this thesis.

These results motivate the development of optimization cost models that minimize cost for measured content request rates as opposed to more complex strategies that include interval caching, in the context of scalable CDNs, which is the topic addressed in the next two chapters.

Chapter 6

Optimal Server Content for Scalable Distribution Networks

The next goal of this thesis is to develop methods for designing CDNs that use one of the recently proposed scalable multicast streaming protocols [69, 82, 83, 111]. In particular, we consider the efficient scalable bandwidth skimming streaming protocol [60, 61]. The focus of this chapter is on determining the proxy content that minimizes total delivery cost for different CDN workloads, assuming a fixed number of proxy servers, fixed (origin and proxy) server placement and that clients send requests either to a fixed proxy or directly to the origin server. The definition of the delivery cost minimized in the CDN design models developed in this chapter is given by equation (2), in chapter 3, which uses approximate network bandwidth costs. In particular, it uses a constant as the ratio of the average cost of a proxy stream to the average cost of an origin stream. These assumptions will be relaxed in the next chapter where the optimal server content, server placement and routing are investigated jointly, using more precise network bandwidth costs that include the cost of network bandwidth over each hop in the delivery tree.

Determining the optimal proxy content for CDNs that use multicast streaming is significantly more complex than for unicast streaming, because of the new tradeoffs introduced by stream sharing, discussed further in section 6.1.1. A key conclusion from previous studies of dynamic skyscraper systems [57, 58] is that storing file prefixes rather than full files and batching client

requests for suffix streams from the origin significantly reduce delivery costs for many workloads and proxy server constraints. However, it is not clear whether the same conclusions hold for the more efficient and flexible bandwidth skimming protocol.

The main goals of this chapter are to: (1) develop extensions to the bandwidth skimming streaming protocol to enable proxy caching of file prefixes, (2) create efficient optimization models that could be used in practice to determine the optimal proxy server content for these protocol extensions, (3) evaluate the effectiveness of alternative protocol extensions, including batching versus non-batching, multicast versus unicast delivery from the origin server, and the impact of the available client bandwidth on the optimal CDN delivery cost, (4) obtain insights into the cost-effectiveness of proxy servers for scalable CDNs and (5) determine the proxy storage strategy (prefix versus full file caching) that (nearly) minimizes the total delivery cost. To achieve these goals, we apply the cost models over a wide region of the design space, including unconstrained as well as realistically constrained proxy storage and disk bandwidth. We assume each proxy uses the scalable bandwidth skimming delivery protocol. For most of the experiments, the origin server also employs the same protocol. However, we also consider the case when multicast is not available over the network from origin to proxy servers, and thus the origin uses simple unicast streaming.

The rest of this chapter is organized as follows. Section 6.1 defines the new extensions to the bandwidth skimming protocol to enable proxy prefix caching and the corresponding delivery cost models for these protocols. The results for CDNs with unconstrained proxy servers are provided in sections 6.2 and 6.3. Section 6.4 provides the results for CDNs with limited proxy resources. Section 6.5 summarizes the chapter.

6.1 New CDN Protocols and Cost Models

This section develops three alternative extensions to the bandwidth skimming protocol and corresponding delivery cost optimization models. The optimization models are based on the previously proposed delivery cost function, given by equation (2) in chapter 3, with new server bandwidth requirements computed for the proposed protocol extensions.

Recall that the required server bandwidth for this protocol, measured in units of the streaming rate, for Poisson arrivals (default assumption in this thesis) and assuming the client requests the whole file is given by $B(N) = \eta \ln (1 + N/\eta)$, where N is the client request rate (as defined in chapter 3) and $\eta = 1.62$ if client has enough bandwidth for receiving two simultaneous streams. We have also verified, through simulations, that the above formula with $\eta = 5.53$ (proposed in [61]) is an accurate estimate for the server bandwidth requirements of bandwidth skimming if client receive bandwidth is equal to 1.2 times the streaming rate. The results obtained with simulation and with the above formula are within 12% of each other.

Section 6.1.1 provides some quantitative examples to illustrate the tradeoffs introduced by scalable delivery. Section 6.1.2 discusses specific system assumptions and parameters to develop the CDN design cost models in this chapter. Sections 6.1.3 – 6.1.5 define three new modifications to the bandwidth skimming protocol to enable prefix caching at proxy servers and derive formulas for the delivery cost model for each of the three new extended protocols, which are called BWSkim(*b*), BWSkim/U(*b*) and BWSkim[/U]+Batch(*b*).

6.1.1 Motivating Examples

Consider a simple scenario where a CDN consists of ten proxy servers where a popular file may be stored and assume that the client request rate N at each proxy for the file is equal to 100. If the file

is not stored at the proxies, all clients must retrieve it from the origin. Thus, in that case, the total request rate at the origin is $N = 1000$. Using the formula for the required server bandwidth for the bandwidth skimming streaming protocol, with $N = 1000$, and assuming clients can listen to two concurrent streams, the required average origin server bandwidth is approximately twelve concurrent streams. However, if the file is fully stored at all proxies, using the same formula with request rate equal to 100, the required average server bandwidth for each proxy is approximately seven streams. Whether ten proxy servers, each transmitting, on average, seven concurrent streams is cheaper than an average of twelve streams from one origin server depends on the ratio of the average cost of an origin stream and the average cost of a proxy stream, and ultimately, on server and network bandwidth costs. Similar tradeoffs also occur for other scalable streaming protocols. For example, periodic broadcast [83, 111] requires a fixed server bandwidth (e.g. in the order of 10 streams) for the origin and for *each* proxy that stores the file, independent of the client request rate.

Provisioning the proxy storage becomes even more complicated if file prefix caching is allowed at the proxy server. As an example, consider the same CDN as in the previous example with 10 proxy servers and client request rate at each proxy equal to 100. If each proxy stores a prefix of the file, say the first 10% of the file, the per-proxy client request rate for the cached content, normalized to the length of the prefix, is 10. Using $N = 10$ in the formula for required server bandwidth for bandwidth skimming, we find that approximately three streams are required at each proxy to deliver the cached prefix. If the prefix is not stored at the proxy, the total client request rate at the origin server for the prefix, normalized to the length of the prefix, is $N = 100$ and, thus, approximately seven origin streams are required to deliver the prefix. As before, whether, an average of seven origin streams is cheaper than an average of three concurrent streams from each of the ten proxy servers depends on the relative cost of an origin stream and a proxy

stream. We point out that, as before, a similar tradeoff also occurs for other scalable protocols. Whether to cache or not a prefix (or a full file) depends on the relative cost of proxy and origin stream and on the ratio of the number of proxy streams to the number of origin streams required to deliver the prefix (or the full file), which ultimately depends on the efficiency of the protocol used.

6.1.2 Model Assumptions and Parameters

The cost models developed in this chapter are based on the general delivery cost formula given in equation (2), in chapter 3. We derive formulas for B_{origin} and B_{proxy} for three classes of CDN protocols, based on extensions to the bandwidth skimming protocol, to determine the proxy content that minimizes the formulation of the delivery cost given in equation (2). To derive these formulas and with the purpose of gaining insights into the optimal CDN server content for different system configurations, we assume a fixed number of proxy servers, fixed origin and proxy server placement, with one proxy located close to each client site (or to a group of client sites) and an origin server located across the "remote" network (as in Figure 1, in chapter 1) and that client requests are sent either to a fixed proxy server or directly to the origin. CDN design models that allow for more flexible server placement and routing are investigated in the next chapter.

Furthermore, for simplicity in deriving the formulas for B_{origin} and B_{proxy} in sections 6.1.3 – 6.1.5, we will assume that the proxy client workloads and proxy storage and bandwidth capacities are statistically homogeneous and we will use the same constant β, defined as the ratio of the average cost of a proxy stream to the average cost of an origin stream, for all proxies. These two last assumptions imply that the same arbitrary fraction f of each file is stored at all proxies. In the experiments in sections 6.2 – 6.4, the value of β will be varied to determine its impact on the optimal proxy storage content. Given these assumptions, the workload and cost model parameters

Table 17: Client Workload and Cost Model Parameters for Defining Optimal Proxy Content for Scalable CDNs

Symbol	Definition
N	Total client request rate, in units of requests per average file playback time
b	Client bandwidth (in units of the file streaming rate)
f	Fraction of the file stored at the proxy
B_{proxy}	Server bandwidth required to deliver the content stored at proxy, measured in units of the file streaming rate
B_{origin}	Server bandwidth required to deliver content from the origin, measured in units of the file streaming rate
β	Ratio of the average cost of one proxy server stream to the average cost of one origin server stream
P	Number of proxy servers

that must be considered to derive the formulas in sections 6.1.3 – 6.1.5, which are taken from the complete list provided in chapter 3, are summarized in Table 17. Note that N is the total client request rate for a file, expressed in number of arrivals per file duration. Thus, N/P is the (homogeneous) client request rate at each proxy. The delivery cost D that is minimized to determine the fraction f of a given file that should be stored at each one of the P proxies, is then restated as:

$$D(f,N,P,b) = B_{origin}(f, N,P,b) + P \times \beta \times B_{proxy}(f, N,P,b). \quad (5)$$

We point out that the formulas presented below can be modified to include different types of heterogeneity. In particular, we have modified them to allow for non-uniform file sizes and streaming rates and heterogeneous client request rates and proxy capacities. Appendix C provides the required modifications to the formulas. Experimenting with these heterogeneous models and extending them to include other types of heterogeneity and, possibly, bandwidth formulas that account for client interactivity are interesting directions for future work.

In sections 6.2 – 6.4, the delivery cost given by (5) will be minimized for a given number of proxy servers and fixed set of client request rates N and access probabilities for each file. We will vary the number of proxies to determine its impact on the minimum delivery cost. In particular, we will evaluate when it is cost-effective to store content at the proxy servers. Furthermore, note that file access frequencies and client request rates may change over time. In fact, for the workloads analyzed in chapter 4, they change significantly, even during a day. In that case, as discussed in section 6.5, the optimal proxy content may need to be recomputed in response to those changes.

6.1.3 BWSkim(*b*) Protocols

This section, together with sections 6.1.4 – 6.1.5, defines three new ways of employing bandwidth skimming in a CDN. They also derive the formulas for B_{origin} and B_{proxy} for each protocol. These formulas were validated against simulation over a wide range of parameter values (N/P = 1 – 10,000, f = 0.01 – 0.99 and P = 1 – 10) and found to be within 15% of the simulation values.

The new CDN delivery protocols are illustrated in Figure 27. The x-axis represents time and the y-axis represents the position on the file for each stream, as the stream progresses. Solid lines represent the basic stream received by each client, i.e., the stream that is created upon arrival of the client request. Dashed lines represent the total amount of data received by each client, which includes the data received in the basic stream and the data received in the target stream. A stream is terminated when it merges with an earlier stream or when it reaches the end of the file.

In the BWSkim(b) protocol, proxy and origin servers use the closest target bandwidth skimming protocol (assuming client bandwidth b), as illustrated in Figure 27(a) for a file with a fraction f stored at the proxy. A new client stream from the proxy is merged hierarchically with other streams from the same proxy. Similarly, multiple origin streams are also merged. The origin

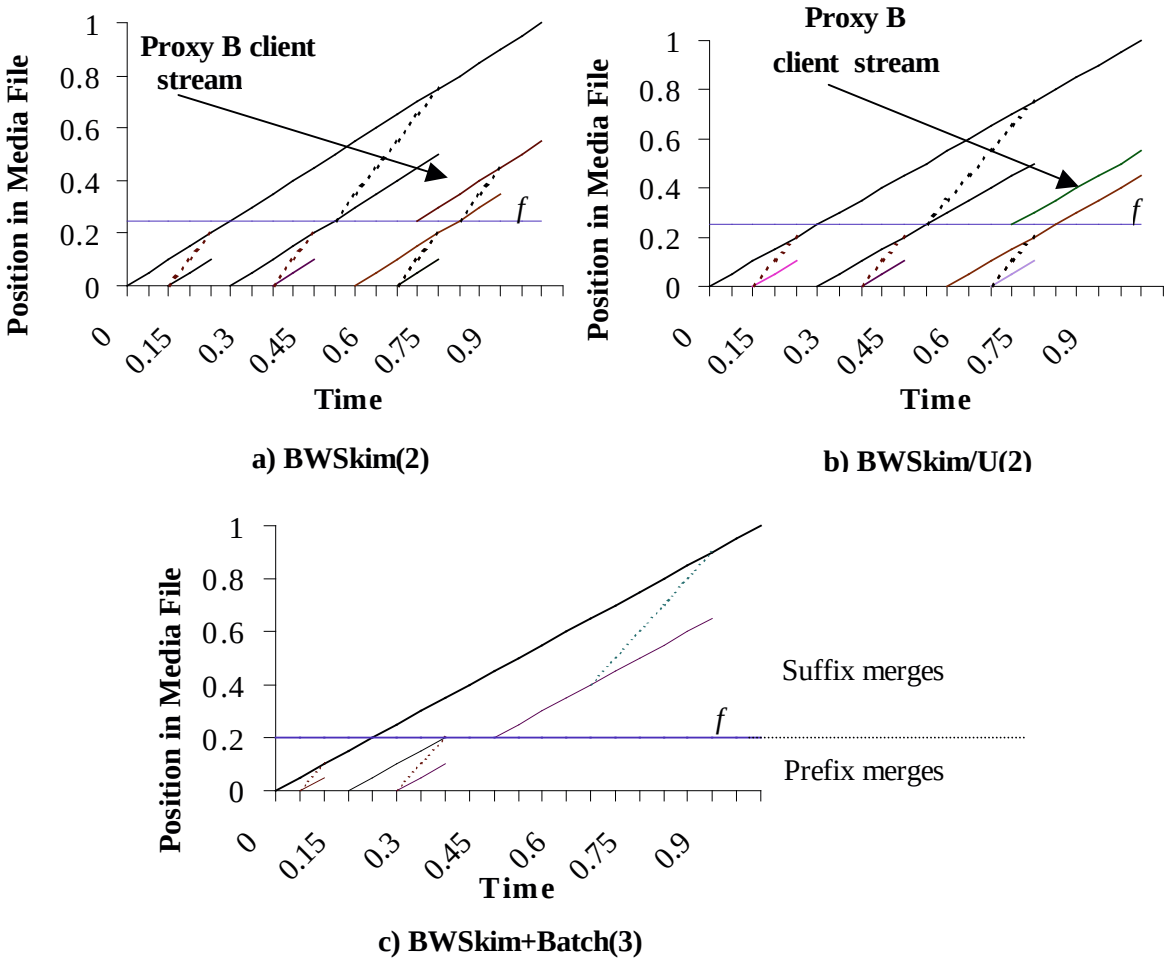


Figure 27: CDN Delivery Protocols (Streams Requested by Proxy A Clients)

may stream the data directly to the clients rather than through the proxies, which only affects the value of β.

If a prefix is stored at the proxies, there are two possible variations of the BWSkim protocol. In the basic implementation, a client listening to the oldest active proxy stream *does not* try to, simultaneously, listen to an origin suffix stream. Rather it waits until the prefix is fully received, and then starts listening to a suffix stream and at the same time tries to catch with the closest target earlier suffix stream. This protocol can be optimized, with a small increase in implementation complexity, by letting a client that is currently listening to the oldest active prefix stream,

simultaneously listen to an origin suffix stream. This optimization requires a few extra messages from origin to proxies, informing them of each new origin stream that delivers content for any file that has a prefix stored at each of them.

In deriving the server bandwidth requirements for BWSkim, we do not use the optimized protocol in order to simplify the calculations. However, we have simulated the optimized protocol and found its required server bandwidth is very close to the formula derived below (usually within 10%). One explanation for this result is that at moderate to high request rate, new requests continuously merge with the oldest proxy stream, delaying the time when it can start a successful merging with an origin stream. At lower arrival rates, there is less opportunity for merging and thus variations in the merging protocol do not have a significant impact.

We now derive the formulas for B_{proxy} and B_{origin} for the BWSkim(b) protocol. Each proxy uses bandwidth skimming to deliver an object of length fT, which has normalized client request rate equal to $\lambda/P \times fT = f\ N/P$. Letting $N_{proxy} = N/P$, and given that the constant η depends on b, as discussed in section 6.1.1, the bandwidth requirements for each proxy server is defined as:

$$B_{proxy}(f,N,P,b) = \eta \ln(1+f\ N_{proxy}/\eta). \quad (6)$$

To derive B_{origin}, we first estimate the request arrival rate for suffixes at the origin server. The request rate from each proxy is equal to the rate at which proxy streams reach the end of the prefix without merging into some earlier stream, which can be derived as:

$$\frac{\lambda_{origin}}{P} = \lim_{\delta T \to 0} \frac{B_{proxy}(f+\delta, N, P, b) - B_{proxy}(f, N, P, b)}{\delta T} = \frac{dB_{proxy}(f, N, P, b)}{d(fT)} = \eta \frac{\lambda / P}{fT\lambda / P + \eta}.$$

The normalized request rate to the origin server suffixes is thus given by $N_{origin} = \lambda_{origin} \times (1-f)T$. We have simulated BWSkim(b) protocol over a wide range of parameter values and found that the

assumption of Poisson arrivals for suffixes at the origin is accurate over the whole parameter space of our validation. Thus, the required origin server bandwidth, computed using equation (1), is defined as:

$$B_{origin}(f,N,P,b) = \eta \ln(1+N_{origin}/\eta), \quad (7)$$

where:

$$N_{origin} = \frac{\eta(1-f)N}{fN/P+\eta}. \quad (8)$$

Sections 6.2 – 6.4 evaluate the BWSkim(*b*) protocol with client receive bandwidth $b = 2$ and $b = 1.2$. The only difference in the calculations of the origin and proxy bandwidth requirements is the value of η, which must be set appropriately for the client receive bandwidth considered.

6.1.4 BWSkim/U(*b*) Protocols

In the BWSkim/U(*b*) protocols, the proxies operate as in BWSkim(*b*), but the origin server uses unicast delivery. The origin streams content to each proxy which then multicasts it, using BWSkim(*b*), to the clients. Thus, all suffix and prefix streams are multicast by the proxies to the clients. Note that an origin stream is terminated when clients listening to the multicast from the proxy merge with a target stream. The operation of the protocol is illustrated in Figure 27(b). It is similar to Figure 27(a), except that origin suffix streams can only merge with other origin suffix streams that are sent to the same proxy.

The required proxy server bandwidth is computed as for BWSkim(*b*), using equation (6). The required origin server bandwidth is computed by calculating the required origin bandwidth for the streams sent to one proxy, using equations (7) and (8), and then multiplying that value by the number of proxies *P*. Thus, for homogeneous proxy servers, in order to analyze the delivery cost and/or the optimal proxy content for protocols that use origin unicast, one needs to consider only

the results for $P = 1$. The total delivery cost for larger number of proxies can be calculated by multiplying the cost for $P = 1$ by the number of proxies. Furthermore, the optimal content is the same for any number of proxies. The total required origin server bandwidth for BWSkim/U(b) is defined as:

$$B_{origin}(f,N,P,b) = P\ \eta\ ln\ [\ 1+(1-f)\ N\ /\ (\ f\ N + \ \eta P)\ ]. \quad (9)$$

Note that if a proxy intercepts all the origin streams, the proxy network i/o bandwidth may be significantly higher than it would be if the suffix streams were delivered directly to the clients. Thus, for BWSkim/U (as well as for BWSkim/U+Batch, described next), the cost of an origin stream must include the cost of the extra proxy network bandwidth. In practical terms, for two CDNs that differ only on whether they use BWSkim (BWSkim+Batch) or BWSkim/U (BWSkim/U+Batch), the value of β is smaller for the latter case. Section 6.4 quantifies the per-proxy network bandwidth needed for delivering origin content using origin unicast streaming.

6.1.5 BWSkim[/U]+Batch(*b*) Protocols

For a file that is fully stored or not stored at all at the proxies, BWSkim+Batch(*b*) operates just like BWSkim(*b*-1), with the same formulas for required origin and proxy server bandwidth. For a file that has a prefix stored at the proxies, BWSkim+Batch(*b*) operates as follows: the client uses one unit of bandwidth to listen to the first origin suffix stream that starts after its request and uses the remaining bandwidth to listen to proxy prefix streams. Thus, for those files, BWSkim+Batch requires client bandwidth $b > 2$. Client requests for objects that are partially cached are batched together for an origin suffix stream. The operation of the protocol is illustrated in Figure 27(c). Since each proxy uses BWSkim(*b*-1) to deliver the prefix to the clients, the required proxy server bandwidth can be calculated using equation (6) with η appropriately set for the value of $b - 1$.

Before deriving the required origin server bandwidth, the origin delivery protocol is clarified. During the first fT of the suffix stream, the origin cannot perform any merging of suffix streams because client bandwidth is often entirely used for listening to one suffix stream and the prefix stream(s). If the suffix is longer than fT, the origin delivers the remaining $(1- 2f)T$ segment of the file using BWSkim(2). Because client requests are batched together for an origin suffix stream, the average time between the creation of two suffix requests, given by $fT + 1/N$, is approximately deterministic, unless f, T or N is small. Hence, the arrival rate at the origin server is given by:

$$\lambda_{origin}(f) = \frac{1}{fT + 1/N}. \tag{10}$$

If the fraction f is greater than or equal to 0.25, there is no merging of origin streams. This is because merging of suffix streams may only start after position $2fT$ of the file (fT of the prefix and fT of the first segment of the suffix) and the minimum inter-arrival time of requests to that last segment of the file at the origin is $fT + 1/N$. Thus, if $f = 0.25$, merging could only start after $T/2$, but the closest target stream would be beyond position $3T/4$ by then. In that case, merging cannot occur before the target stream ends.

If the fraction f is less than 0.25, merging may occur in the last $1–2f$ of the file. Since no merging occurs before this last segment, the rate at which "requests" for this last segment arrive is the same (approximately) deterministic arrival rate at the origin server, given by equation (10). The origin bandwidth needed to deliver this last segment is derived (in appendix B), as follows:

$$N^*_{origin}(f) = \frac{1}{fT + 1/N}(1-2f)T,$$

$$k(f) = \lfloor \log_2(N^*_{origin}(f) + 2 \rfloor - 1,$$

$$B_{origin,last\_segment}(f) = \frac{3}{2}k(f) + \frac{N^*_{origin} + 2}{2^{k(f)}} - 2.$$

Therefore, the required origin server bandwidth for BWSkim+Batch(b) is defined as:

$f \geq 0.25$: $B_{origin}(f,N,P,b) = \lambda_{origin} \times (1-f)T$ (11)

$f < 0.25$: $B_{origin}(f,N,P,b) = \lambda_{origin} \times fT + B_{origin,last\_segment}(f)$ (12)

Equations (11) and (12) can be changed for the BWSkim+Batch/U(b) protocol, as shown in section 6.1.4 for BWSkim(b).

For the BWSkim+Batch[/U] protocols, the following sections consider systems with client bandwidth b = 3 and b = 2.2, which implies client bandwidth of 2 and 1.2, respectively, for merging proxy streams. We point out that, for all the protocols proposed, but especially for the BWSkim+Batch protocol, the "client" may actually be a system very close to the real client that buffers and delivers the data in a single stream to it, if the "last mile" bandwidth to the actual client is limited. In any case, BWSkim+Batch has higher implementation complexity than BWSkim, because it requires more synchronization between proxies and origin for allowing batching of requests. Therefore, it is preferable to use BWSkim unless the cost savings for BWSkim+Batch are significant. Of particular interest are the relative performance of BWSkim+Batch(b) with b = 3 and b = 2.2 compared to the simpler BWSkim(2) and the cost increase for BWSkim(1.2), which may be important for streaming higher quality content over a non-broadcast network.

6.2 Storage Content for Unconstrained Proxy Servers

This section analyzes the optimal delivery protocol and storage strategy for the case where the CDN proxy servers can be configured with enough storage and bandwidth for the fraction of each file that minimizes the file delivery cost. For each protocol and a wide range of parameters values,

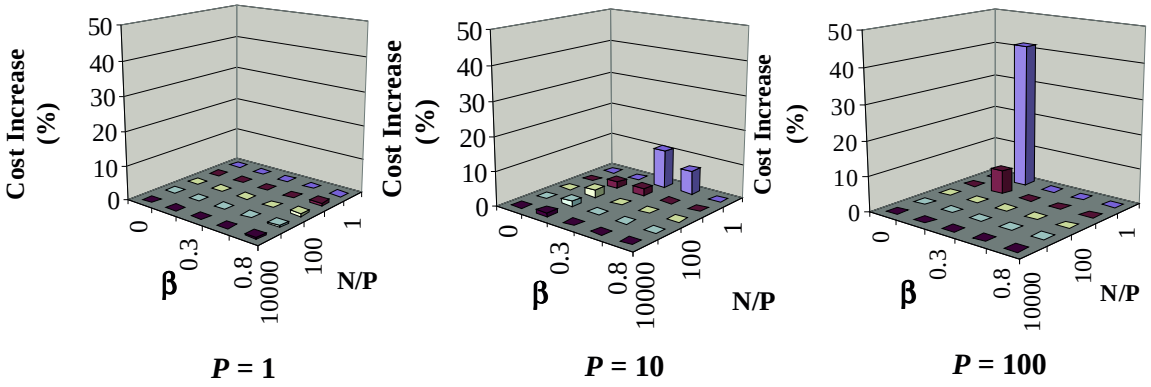


Figure 28: Cost Increase of BWSkim[/U](2) versus BWSkim[/U]+Batch(3)
(unconstrained proxy servers, one file)

we numerically solve the total delivery cost, given by equation (5), for the fraction f of the file stored at the proxy that minimizes the total delivery cost. We experiment with per proxy request rate N/P varying from 1 to 10,000, number of proxies P equal to 1, 10 and 100, relative cost of proxy stream β, varying from 0 – 0.8 and two different values of client bandwidth b for each protocol. We point out that the results in this section provide general insights for provisioning CDNs for multicast streaming, which are key to understand the results for proxies with constrained storage and disk bandwidth, reported in section 6.4.

Figure 28 shows the percent increase in cost of BWSkim(2) compared with BWSkim+Batch(3). Note that the results for $P = 1$ also provide the cost increase for BWSkim/U(2) versus BWSkim+Batch/U(3). The simpler BWSkim(2) has total delivery cost within 5% of the cost of BWSkim+Batch(3) over almost the entire design space (except for $P > 10$ and small N/P). Furthermore BWSkim/U(2) performs just as well as BWSkim+Batch/U(3).

The optimal fraction stored at the proxies for BWSkim(2) protocol for the wide design space explored is shown in Figure 29. One key result is that the optimal value of f is either 1 or 0,

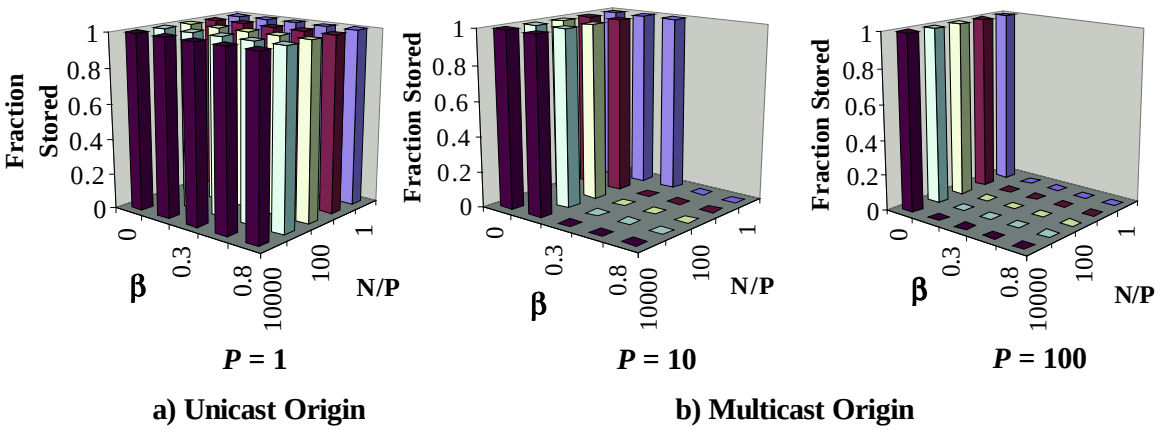
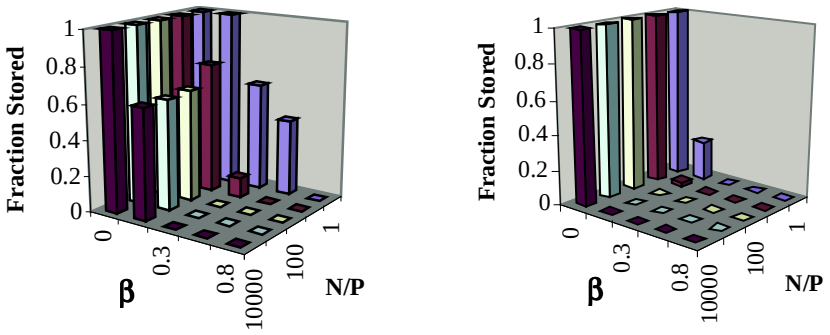


Figure 29: Optimal Proxy Content for BWSkim[/U](2)
(unconstrained proxy servers, one file)

depending primarily on P and β, and to a lesser extent on N/P. Whereas Figure 29(a) shows that the file should always be stored at the proxies if $P = 1$ (or the origin uses unicast), Figures 29(b) and 29(c) show that, if $P > 1$ (or the origin uses multicast), the file should only be stored at the proxies if the total request rate N is small and β is small. In other words, the cost of a proxy stream must be significantly lower than cost of an origin stream to compensate for the less sharing available for proxy streams, compared to the sharing available for origin streams if all clients retrieve the file from the origin. Note that, in this case, the assumption of unconstrained proxy resources may be reasonable. Note also that an all-or-nothing caching policy, which stores either the full file or no file at all, is much simpler to implement than a caching policy that stores file prefixes of variable length.

In contrast with the results in Figure 29, Figure 30 shows that the optimal proxy content for BWSkim+Batch(3) for $P = 10$ and $P = 100$ includes file prefixes in several regions of the design space. Most importantly, for the parameter values where BWSkim+Batch outperforms BWSkim, BWSkim+Batch caches a file prefix. We point out that BWSkim+Batch(2.2) is even less cost-



a) *P* = 10 **b) *P* = 100**

Figure 30: Optimal Proxy Content for BWSkim+Batch(3)
(unconstrained proxy servers, one file, multicast origin)

effective, compared to BWSkim(2), than BWSkim+Batch(3). Therefore, for unconstrained proxy servers, BWSkim(2) is always preferred over BWSkim+Batch(2.2).

We now discuss the significance of the optimal solution, computed using the design models, for a scalable CDN. Firtst, we point out that the difference, in terms of the total delivery cost, between a CDN that uses the preferred protocol and stores the optimal content and a CDN that uses a less preferred protocol and stores non-optimal content can be very significant. Figure 28 shows that it can be as high as 40% for the cases when BWSkim+Batch(3) is the optimal protocol, in which case the optimal proxy content is a file prefix, and BWSkim(2) is used. Furthermore, even in the cases where BWSkim(2) is the preferred protocol, storing non-optimal content at the proxies (i.e., full file when the optimal content is no file at all or vice versa) can significantly increase total delivery cost (up to 200% in some cases), as will be shown in the next section.

The optimal all-or-nothing storage results for CDNs that use BWSkim(2) contrast sharply with results in previous works on unconstrained proxy storage strategies for CDNs that use either the scalable Partitioned Dynamic Skyscraper (PDS) [57] or variations of the patching protocol [118]. In [118], the authors propose the use of prefix caching and batching but do not compare it against

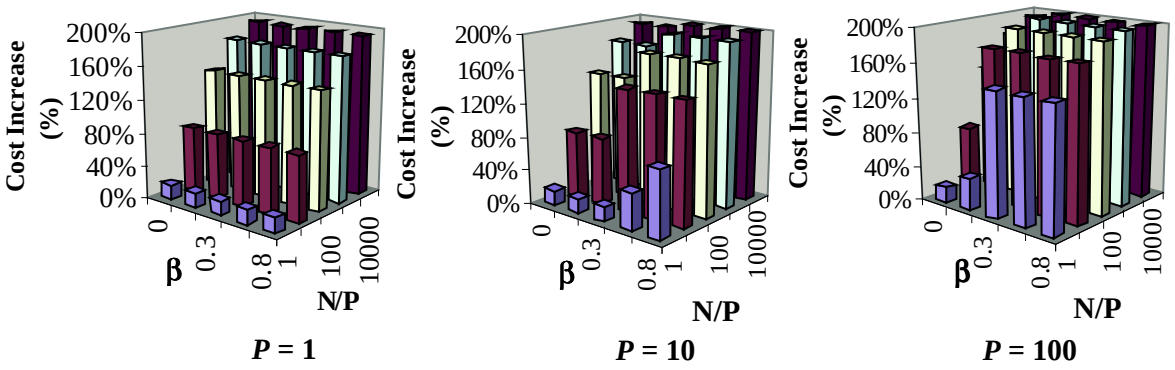


Figure 31: Cost Increase of BWSkim[/U](1.2) Compared to BWSkim[/U](2)
(unconstrained proxy servers, one file)

streaming the whole file from only one server. Our results show that, in a BWSkim(*b*) system, which is based on a more efficient streaming protocol than the PDS protocol, batching of clients for origin streams is not more effective than the very efficient bandwidth skimming merges that occur in the streams delivered by the proxies or by the origin.

Figure 31 shows the percent increase in delivery cost for BWSkim(1.2), compared to BWSkim(2). The optimal proxy storage is the same for both policies in nearly every point of the design space. Thus, the cost difference is due to the lower merging efficiency for smaller values of *b*. As *N*/*P* increases, the cost increase converges to 200%, which implies that the optimal delivery cost increases by at most a factor of three if client bandwidth is limited to 1.2 streams (i.e., for higher quality videos).

Based on these results, we draw the following conclusions for provisioning CDNs based on unconstrained proxy servers with an optimal delivery protocol and a storage policy:

- BWSkim+Batch(3) and file prefix caching are the most cost-effective strategies *only* if the origin uses multicast delivery, the number of proxies is large, arrival rate at each proxy is low and for intermediate values of β (i.e., $P > 10$, $N/P \approx 1$ and $\beta = 0.1$, as in Figure 28(c)). In all

other regions of the design space, the simpler BWSkim(2) and all-or-nothing storage strategies are (nearly) optimal.

- In a bandwidth skimming CDN, a given file (or file prefix) is stored at the proxies only if: (a) the origin uses unicast (or $P = 1$) or (b) $\beta = 0$ or (c) β, P and N/P are small. Furthermore, if the origin uses multicast, the larger the value of P, the lower the value of β must be so that it is cost-effective to store content at the proxies, (e.g. $\beta \leq 0.5$ for $P \leq 10$ or $\beta \leq 0.1$ for $P > 10$). The cost-effectiveness of storing content at proxies is further discussed in the next section.
- BWSkim(1.2) increases the delivery cost by up to 200%, which might be reasonable, given it allows the delivery of higher quality videos to the clients.

6.3 Clarifying the Cost-Effectiveness of Proxy Servers in Scalable CDNs

The results in the previous section showed that, if origin uses unicast, replicating content at multicast-enabled proxies is always cost-effective in reducing bandwidth requirements. However, if the origin uses multicast, storing a file or a file prefix at the proxies is only cost-effective if the cost of a proxy stream is much lower than the cost of an origin stream (i.e, β is small) or the total request rate N is small. If N is small, the limited opportunity for sharing of origin streams makes it advantageous to store content at the proxies. However, if N is very small, multicast is not very effective. As N or β increases, storing content at the proxies becomes less attractive as it is more cost-effective to let all clients share an origin stream. Note that a very small prefix might be stored at the proxies or at the clients in order to reduce start-up latency or for smoothing VBR streams.

Figure 32 shows further data to clarify how small β must be for proxy caching to be effective. The total delivery cost is plotted as a function of the number of proxies, for given values of β and total arrival rate N, using the optimal protocol at each point and assuming the origin uses multicast.

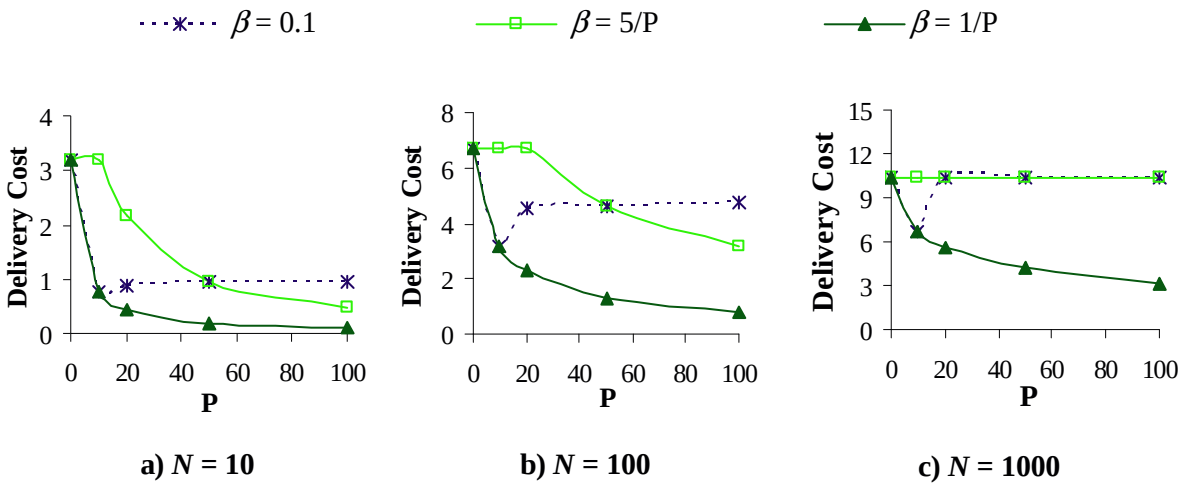


Figure 32: Delivery Cost Versus Number of Proxies (*P*) for Optimal Scalable Multicast Protocol (unconstrained proxy servers, one file)

Unless β is a very smallfraction (approximately 1/*P* or less), adding extra proxy servers does not improve the total delivery cost. For instance, take the curves for β = 0.1 and note that the total delivery cost decreases only up to *P* = 10. Furthermore, given a number of proxies, the total delivery cost is also lower for smaller values of *N*/*P*. Given these results, the experiments in section 6.4 for proxy servers with constrained disk storage and bandwidth consider only $\beta \leq 0.1$, if the origin server uses multicast.

Figure 32 can also be used to illustrate the significance of the optimal proxy storage strategy for a given protocol. In particular, it can be used to determine the increase in the total delivery cost, compared to the optimal, if a file is not stored at the proxy when the optimal solution dictates it should be, for the cases when BWSkim(*b*) is the preferred protocol. Note that the curve for β =1/*P*, with decreasing delivery cost as a function of the number of proxies, consists of cases when the optimal proxy content for BWSkim(*b*) is the full file. Recall that for β = 0.1 and large values of *N* and *P*, it is more cost-effective not to store the file at the proxies, as shown in Figure 29. If the file

is not stored at the proxies, the total delivery cost obtained with $\beta = 0.1$ is the same as the total delivery cost obtained with any value of β (in particular, $\beta = 1/P$), since the required proxy server bandwidth B_{proxy} is equal to 0. Therefore, considering the cases when BWSkim(b) is the preferred protocol, for the regions of the design space where the optimal proxy strategy is not to cache the full file when $\beta = 0.1$ (i.e. $N = 1000$ and $P > 10$), the difference in the total delivery cost for $\beta = 1/P$ and $\beta = 0.1$ corresponds to the difference in the total delivery cost if the non-optimal storage strategy is used compared to the minimum delivery cost (obtained with the optimal proxy content). As shown in Figure 32(c), the increase in total delivery cost can be very significant (up to 200%). A similar approach can be used to show that, for the cases when BWSkim+Batch(b) is the preferred protocol, using a non-optimal proxy storage strategy can be significantly sub-optimal.

6.4 Storage Content for Constrained Proxy Servers

This section uses the cost models to provision a CDN that has *limited* proxy storage capacity and disk bandwidth with a delivery protocol and storage policy that (nearly) minimizes the total delivery cost defined in equation (5). Considering a set of origin media files, each with a different access frequency, the main goal is to obtain insights into which fraction of each file should be stored at the proxies and which delivery protocol has higher performance in the scenarios, identified in sections 6.2 and 6.3, where storing content at the proxies is cost-effective, i.e., if the origin server uses unicast or $\beta \leq 0.1$ (for $P \leq 100$).

The optimal delivery protocol and storage strategy is obtained by solving the optimization outlined in section 6.1 for each protocol and set of parameter values, with constraints on the required proxy bandwidth (B_{proxy}) and required storage space, defined as the sum, over all files, of the fraction of the file cached at the proxy times the file length. Simple storage strategies that

Table 18: Additional Parameters for Determining Optimal Content for Constrained Proxy Servers in Scalable CDNs

Symbol	Definition
T	Media file duration (in minutes)
n	Number of media files, each of duration T
p_i	Access probability for file i (determined by the Zipf-like distribution)
M	Total client arrival rate, in arrivals per T (all files)
C	Proxy storage capacity as a fraction of n×file size
B	Ratio of available proxy disk bandwidth to the average bandwidth required if the proxy fully stores all n files

achieve nearly minimum cost are preferred over more complex, but truly optimal strategies. In particular, if the optimal solution includes a set of objects that must be enumerated, we further constrain the models to obtain a near optimal but simpler solution where the content can be specified by a simple range of relative access frequency and then compare the cost of both solutions. Additional parameters and modifications to the models to guarantee global optimality are discussed in sections 6.4.1 and 6.4.2. Relevant results for CDNs where the origin uses unicast and multicast delivery are summarized in sections 6.4.3 and 6.4.4, respectively.

6.4.1 Additional Parameters

The additional system and workload parameters needed for applying the models to CDNs with constrained proxy resources are shown in Table 18. We consider a set of n files of equal duration T, with total request rate (for all files) equal to M[4], expressed as the unit of arrivals per time T, and file access frequency given by a Zipf distribution with parameter θ for all proxy servers. As in chapter 5, we will vary n and set the value of $\theta = 0$. The available proxy storage C is normalized and

[4] Note that we use the symbol M for the total request rate for all files whereas N was used in the previous sections to represent the total request rate for a given file.

expressed as a fraction of the total amount of data at the origin, as in chapter 5. The available proxy bandwidth is also normalized but the normalization process is different than the one used in chapter 5. The available proxy bandwidth, *B*, is expressed as a fraction of $r \times BW_2^*$, where *r* is the average file streaming rate and BW_2^* is the average number of active streams that would be needed if the proxy fully stored *all* files. Given p_i, the probability of an access for file *i*, determined by Zipf distribution, and *M*/*P*, the total arrival rate at each proxy, BW_2^* is calculated, using equation (2), as follows:

$$BW_2^*(M, p_i) = \sum_{i=1}^{n} \eta \log(1 + \frac{M/P \times p_i}{\eta}). \quad (13)$$

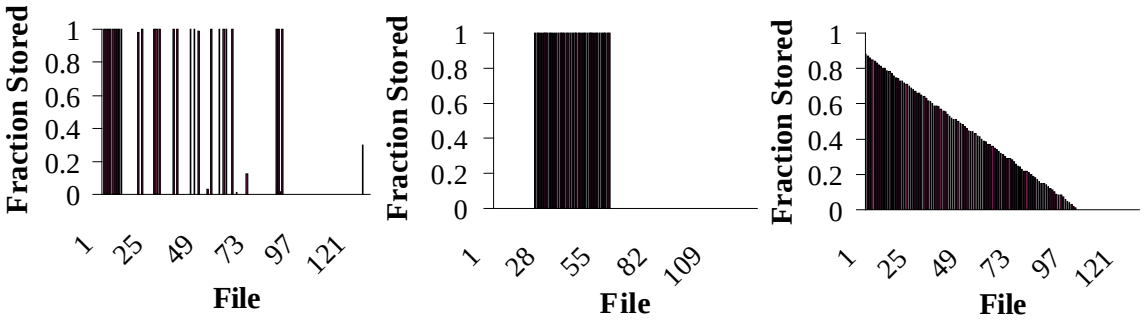
To obtain realistic relative values of *C* and *B*, for given values of *M*/*P*, *T* and *n*, we use an integral number of Ultrastar 72zX disks and MPEG-2 streaming rate of 4Mb/s. One such disk can hold approximately 44 hours and can sustain 42 concurrent streams of MPEG-2 content (Table 2 in chapter 3). Fixing *C*, *M*/*P* and *n*, shorter (longer) duration files will need a smaller (larger) number of disks. Thus, for a given value of *C*, the value of *B* will be smaller for shorter videos and larger for longer videos. In other words, by varying *n* and *T*, the relative difference between *C* and *B* is varied. In order to explore a wide range of values of *C* and *B*, we choose *n*, the number of files to be either 128 or 1024, and the file duration *T* to be either 30 or 120 minutes and vary the number of disks so that *C* varies from 0.08 to 0.68. Furthermore, we vary *M*/*P* from 10 to 10000 so that *B* varies from 0.02 to values greater than 1. As a result, a wide range of absolute and relative values of *C* and *B* are included in the experiments.

6.4.2 Linear Cost Models

The cost models developed in section 6.1 are based on sums of logarithms, which are non-linear non-convex functions, for which the algorithms for solving the optimization problem are not guaranteed to find a global optimal solution. In order to guarantee true optimal solutions, the logarithm functions are replaced by an expression that is based only on linear functions and binary variables. We use discrete values of f, ranging from 0 to 1, in steps of 0.01, We pre-compute the bandwidths for each possible value of f and store them in a table. Binary variables θ_k are introduced to indicate whether the fraction stored at the proxies is equal to the fraction stored in entry k of the table. The required origin and proxy server bandwidths are each redefined as a linear function of the variables θ_k, where each binary variable θ_k is multiplied by the pre-calculated bandwidth stored in entry k of the table. Note that only one variable θ_k can be equal to 1, as it indicates which fraction f of the file must be stored at the proxies. The modified model is a mixed integer problem which can be solved to true optimality using the efficient CPLEX solver, available as part of the GAMS system [27].

6.4.3 Storage Content If Origin Server Uses Unicast Streams

If the origin uses unicast, the proxy content that minimizes delivery cost does not depend on P. Furthermore, as β increases, the relative cost of a proxy stream increases, but the proxy content that minimizes delivery cost for either BWSkim/U or BWSkim+Batch/U remains the same. Since suffix streams sent to clients of different proxies do not merge, the merging efficiency is the same regardless of the value of β. Thus, there is no motivation for storing different content as β changes. The results shown below as well as in the next section are marked with the keywords “cap”, “bw”



a) True Optimal
B = 0.32 (bw)

b) Near Optimal
B = 0.32 (bw)

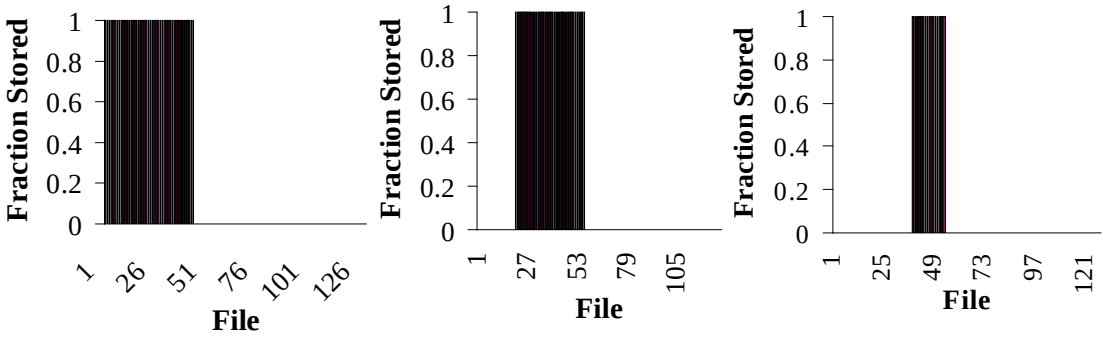
c) B = 1
(unrealistic B)

Figure 33: Example Proxy Content for BWSkim/U(2)
(constrained proxy servers, unicast origin, C = 34%, M/P = 1000, n = 128, T = 2h)

or "both" to indicate when the optimal proxy content uses all the proxy storage capacity or proxy bandwidth or both, respectively.

The optimal proxy content for BWSkim/U(2) consists primarily of full files, but for many configurations, the optimal content is a complex set of full files and a few file prefixes, as illustrated in Figure 33(a). In all such cases, the near optimal solutions constrained to be any contiguous (with respect to their relative access frequency) set of full files, illustrated in Figure 33(b) for the same configuration of Figure 33(a), yield total delivery cost within only 0.05% of the minimum. Thus, these near optimal solutions are presented in the remainder of this section.

The (near) optimal all-or-nothing storage strategy for BWSkim/U(2) is significantly different from the results obtained in parallel studies which show that prefix caching is the optimal storage strategy [76, 144, 147] for CDNs with constrained proxy storage capacity but unlimited disk bandwidth. In order to understand the reason for this discrepancy, we solved the model for the same configuration used in Figures 33(a) and 33(b) (in particular, for the same values of C and M/P) but with proxy bandwidth, B, arbitrarily set to 1. Note that this value is three times larger the actual (realistic) value of B calculated based on the values of the other parameters and that



a) ***B*** **≥ 1,** ***C*** **= 34%**
(cap) T = 2h
M/P = 100, n = 128
M/P = 1000, n = 1024

b) ***B*** **= 0.32,** ***C*** **= 34%**
(bw) T = 2h
M/P = 1000, n = 128
M/P = 10000, n = 1024

c) ***B*** **=0.13,** ***C*** **= 34%**
(bw) T = 2h
M/P = 10000, n = 128

Figure 34: Optimal Proxy Content for BWSkim/U(2)
(constrained proxy servers, unicast origin)

arbitrarily setting B to 1 is the same as removing the constraint on the proxy bandwidth. Figure 33(c) shows that the optimal content (not constrained to be a contiguous set of full files) for this hypothetical scenario consists only of file prefixes, in agreement with the parallel results (in particular these results are very similar to the results obtained in [147] for a similar CDN configuration that uses the patching protocol). However, this configuration is not feasible for the expected disk technology trends and for the given request rate and number of files. For feasible configurations, our results show that with the same value of C and B used in Figure 33(c) (but a lower request rate M/P), the optimal proxy content does not include prefix caching (Figure 34(a)).

Over the wide range of system parameters explored, we found that the delivery cost savings for BWSkim+Batch/U(3) compared to BWSkim/U(2) are very small (i.e., lower than 8%). Furthermore, we found that BWSkim+Batch/U(3) stores mainly prefixes at the proxies [11]. However, batching is restricted to clients in the same region and thus, is not very beneficial. Thus, if the origin uses unicast, the simple BWSkim/U(b) and all-or-nothing storage are the most cost-effective strategies.

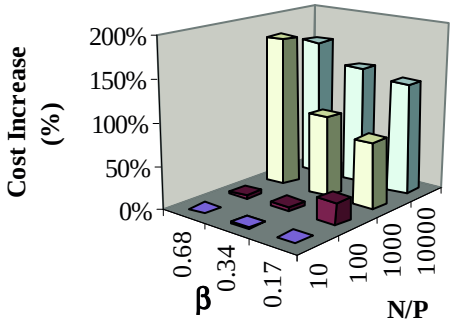


Figure 35: Cost Increase of BWSkim/U(1.2) Compared to BWSkim/U(2)

(constrained proxy servers, unicast origin, n = 128, T = 2h)

Figures 34(a)-(c) show the main trend in the optimal storage content for BWSkim/U(2) as B changes. For values of $B \geq 1$, the proxy fully stores as many of the most popular files as possible. However, as B decreases and the system becomes bandwidth constrained, the proxy fully stores fewer and less popular files.

Figure 35 shows that if client bandwidth is limited (as may be the case for streaming high quality content over non-broadcast networks), using BWSkim/U(1.2) increases the total delivery cost by up to 200%. The trends in the optimal proxy content for BWSkim/U(1.2) as a function of the parameter values are similar to the trends for BWSkim/U(2). Note that the results in Figures 34 and 35 are very similar to the conclusions reached for unconstrained proxy servers.

In BWSkim/U CDNs, the origin uses unicast to stream its content through the proxy to the clients. Therefore, a natural concern in such systems is whether the required proxy network i/o bandwidth is significantly higher than the proxy disk bandwidth (used to stream content that is stored locally). We found that if the proxies are not severely under-provisioned, in terms of storage capacity and disk bandwidth for the client load, the total network i/o bandwidth needed per proxy for streaming origin and local content is less than twice the per-proxy disk bandwidth.

6.4.4 Storage Content If Origin Server Uses Multicast Streams

For CDNs in which the origin uses unicast streams, the results in the previous section show that the simpler BWSkim/U(*b*) delivery protocol and an all-or-nothing storage policy yield (near) optimal delivery cost. Considering CDNs where the origin uses multicast streams, this section evaluates whether (and in which regions of the design space) prefix caching and batching clients from different regions together for origin suffix streams reduce cost significantly.

Figure 36 shows that BWSkim+Batch(3) significantly reduces delivery cost (e.g., by more than 20%) primarily when $\beta \leq 0.1$, $P > 10$, $M/P \geq 100$ and $B > 0.1$. BWSkim(2) is preferred in all other regions of the design space. If M/P is small, multicast is not very effective and the optimal proxy content for both protocols is approximately the same (long prefixes or full file caching of the most popular files). For very large M/P, B is very small (as illustrated in the top of Figure 36), the system is very constrained on bandwidth and, consequently, the optimal proxy content includes fewer and less popular data for both protocols. We point out that these results are consistent with the results found for unconstrained proxy servers. In section 6.2, we showed that BWSkim+Batch(3) only significantly outperforms BWSkim(2) for small β, $P > 10$ and N, the *per file* request rate, approximately equal to 1. Note that M/P is the total request rate for all files. Given the Zipf distribution of access frequency, the per file request rate is very small for many files, for the intermediate values of M/P identified above.

We also found that, as shown in Figure 37, BWSkim+Batch(2.2) reduces delivery cost over BWSkim(2) for the same configurations as BWSkim+Batch(3) and, perhaps surprisingly, by nearly the same amount, in spite of the reduced merging efficiency for prefix streams. Moreover, the

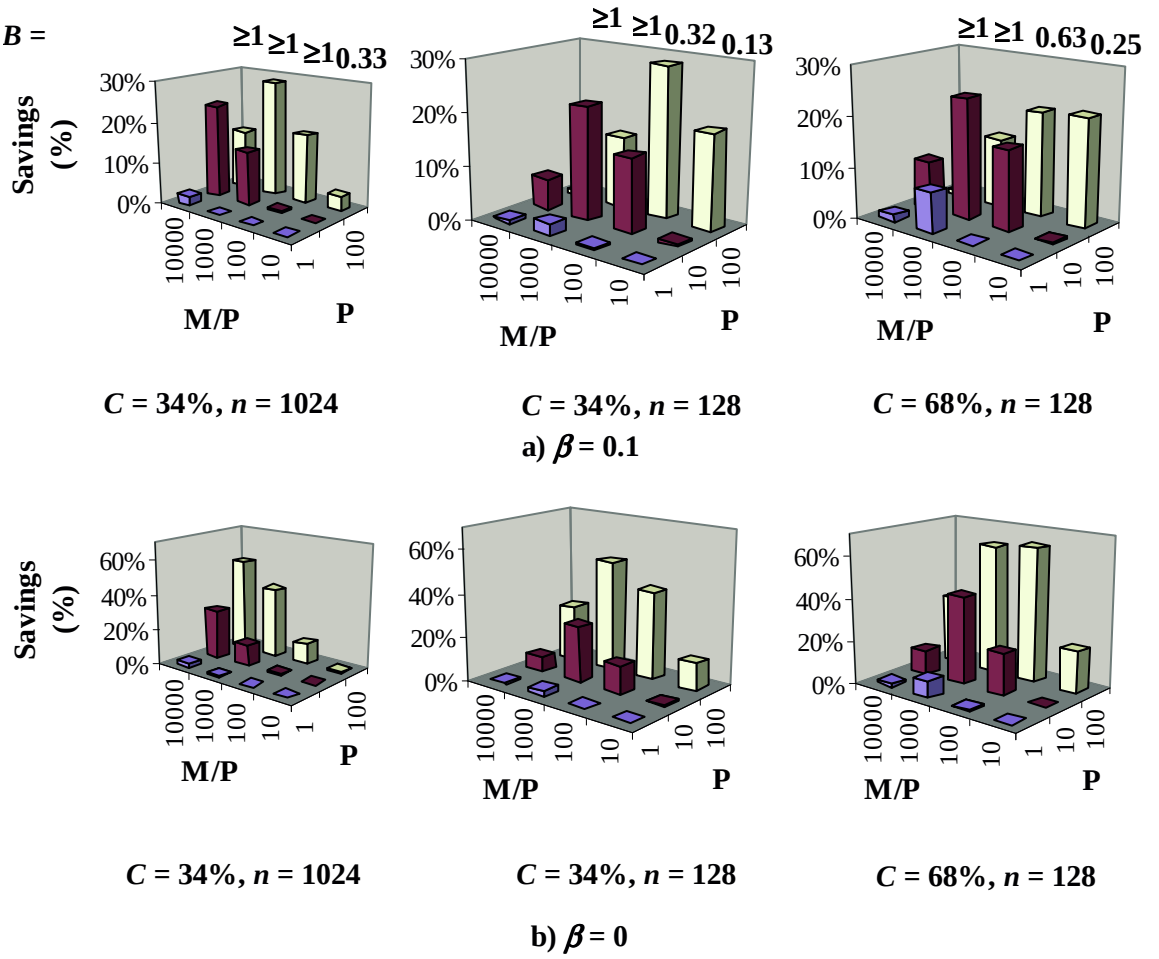
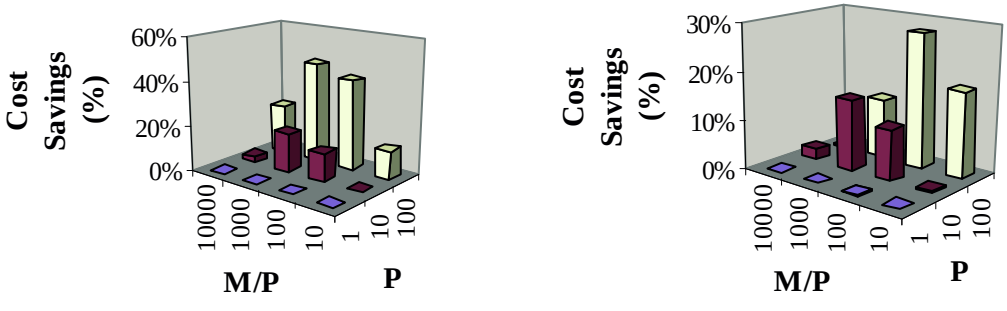


Figure 36: Cost Savings of BWSkim+Batch(3) Compared to BWSkim(2)
(constrained proxy servers, multicast origin, T = 2h)

optimal proxy content for BWSkim+Batch(b) is roughly the same for b = 2.2 and b = 3, for almost all configurations.

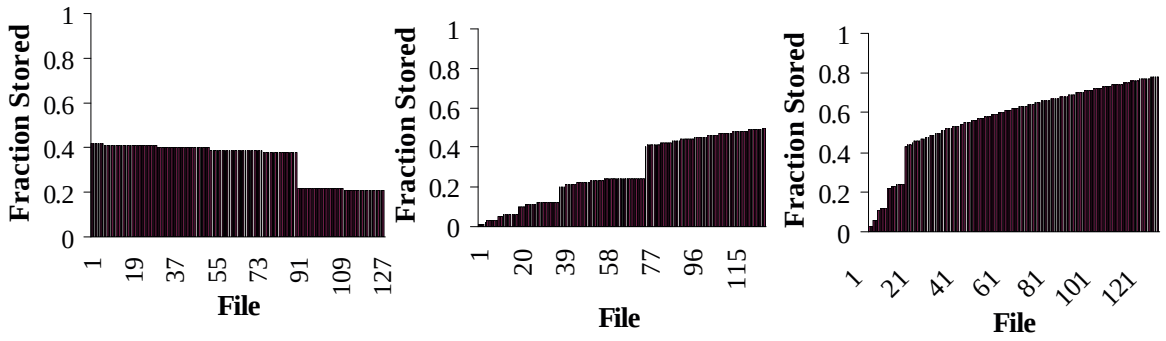
Figures 38 and 39 show how the optimal proxy content for BWSkim+Batch(3) and BWSkim(2) varies, respectively, over the design regions in which each is the best policy. As is the case for CDNs where the origin uses unicast, the optimal proxy content for both protocols consists of less popular and fewer files as B decreases (Figures 38(a)-(b) and Figures 39(a)-(b)). We also observed the same trend as either β or P increases. It is more cost-effective to let clients retrieve



a) $\beta = 0$ **b) $\beta = 0.1$**

Figure 37: Cost Savings of BWSkim+Batch(2.2) Compared to BWSkim(2)

(constrained proxy servers, multicast origin, C=34%, n = 1024, T = 2h)



a) $B \geq 1$, C = 34%, (cap) **b) B = 0.32, C = 34%, (bw)** **c) B = 0.64, C = 68%, (bw)**

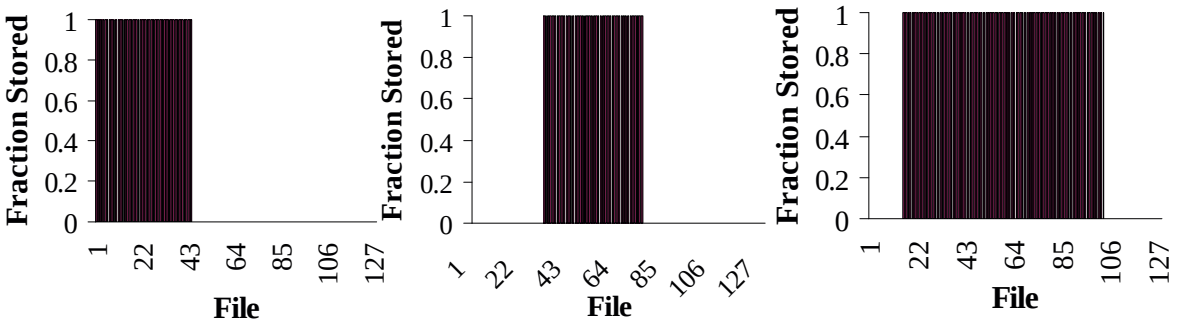
M/P=100 M/P=1000 M/P=1000

Figure 38: Optimal Proxy Content for BWSkim+Batch(3)

(constrained proxy servers, multicast origin, β = 0, P = 100, n = 128, T = 2h)

most of the content (in particular the most popular content) from the origin as either β or P increases, consistently with the results for CDNs with unconstrained proxies. Furthermore, if more disks are added to the proxies, BWSkim stores more files (Figure 39(c)) and BWSkim+Batch stores longer prefixes (Figure 38(c)).

Figure 40 shows that, compared to the best strategy for $b \geq 2$, using BWSkim(1.2) may increase the delivery cost significantly. The larger the proxy storage capacity, the more significant

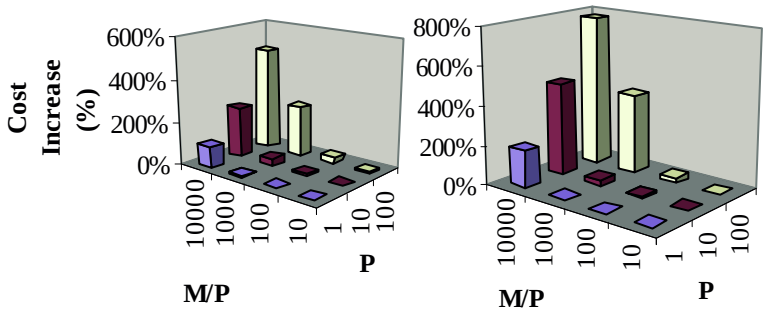


a) $B \geq 1$, $C = 34\%$, **(cap)** **b)** $B = 0.32$, $C = 34\%$, **(both)** **c)** $B = 0.64$, $C = 68\%$, **(both)**

M/P=100 M/P=1000 M/P=1000

Figure 39: Optimal Proxy Content for BWSkim(2)

(constrained proxy servers, multicast origin, β = 0.1, P = 10, n = 128, T = 2h)



a) $C = 34\%$ **b)** $C = 68\%$

Figure 40: Cost Increase for BWSkim(1.2) Over Best Policy when b ≥ 2

(constrained proxies, multicast origin, β = 0, n=1024, T =2h)

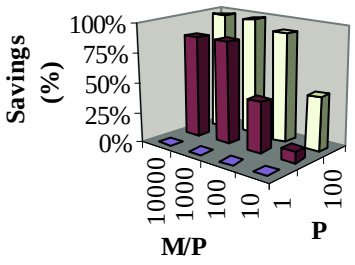


Figure 41: Cost Savings for Multicast Compared to Unicast From Origin

(constrained proxies, β = 0, n = 128, T = 2h)

is the cost increase, because of the more limited merging efficiency of proxy streams for BWSkim(1.2).

We also compute how much savings are achieved when the origin server uses multicast, as compared to unicast. Figure 41 shows some representative results for configurations in which the proxies are provisioned to handle a reasonable fraction of the origin load. These results show that unless the request rate at each proxy (M/P) is small or P = 1, the cost benefit of origin multicast is

very significant. For a CDN with P = 10 proxy servers and overall request rate at each proxy M/P = 1000, the bandwidth savings can be as high as 85%. This observation should encourage a wider deployment of IP or application level multicast protocols in the Internet.

Finally, we point out that the results provided in this section and in section 6.2, though seemingly in conflict, are consistent with results from previous and parallel works. In the previous section, we show that if proxy disk space is limited but proxy disk bandwidth is unrealistically assumed to be unlimited (as in [76, 144, 147]), the optimal storage strategy consists of file prefixes. Furthermore, in the regions of the design space where batching is beneficial, the optimal storage content includes file prefixes. This is consistent with results reported in [118, 143, 144] which shows that prefix caching is optimal for a delivery protocol that uses batching of client requests for origin streams. However, unlike previous study, we also show that there is a vast region of the design space where batching and prefix caching are *not* effective in reducing the total delivery cost. In that region, the much simpler BWSkim(b) and all-or-nothing storage strategies are (nearly) optimal.

6.5 Summary of the Chapter

This chapter proposed three new alternative modifications to the scalable bandwidth skimming streaming protocol to enable proxy caching of file prefixes and developed efficient CDN design models that are applied to derive important insights into the cost-effectiveness of storing content at proxy servers and into the optimal proxy content, over a much wider and more realistic CDN configuration space than previous work.

One key conclusion from this study is that storing content at multicast-enabled proxy servers is only cost-effective if: (a) the origin server uses unicast *or* (b) the file request rate is low (in

which case, multicast is not highly effective) *or* (c) the average cost of a proxy stream is a small fraction (1/*P*, or less, where *P* is the number of proxies) of the average cost of an origin stream, where the cost includes network and server costs. This is in sharp contrast with unicast CDNs where storing content at proxy servers is always beneficial. We point out that, in the cases where proxy caching is cost-effective and if proxy storage and bandwidth are constrained, the optimization models developed in section 6.1 are sufficiently simple and efficient to be applied on line. For example, the solution time for the BWSkim(b), the optimal protocol for a large region of the CDN configuration space, or BWSkim/U(b) is on the order of a few minutes or less, for each of the experiments in section 6.4.

By solving the optimization cost models over a wide CDN configuration space, we found that the more complex BWSkim+Batch(*b*) protocol significantly outperforms BWSkim(b) only if: (a) the origin uses multicast delivery, (b) the number of proxies is very large *and* (c) the per file request rate is small ($N/P \approx 1$). Furthermore, in these limited regions of the design space where batching is beneficial, prefix caching is the optimal storage strategy. However, in the rest of the design space, the simpler BWSkim(b) without batching and an all-or-nothing storage policy (nearly) minimizes delivery cost. This is in contrast to the previous result for the less efficient dynamic skyscraper systems [57, 58], in which batching and prefix caching is often optimal. Moreover, we found that, unlike conventional CDNs, where the optimal proxy content consists of the most popular files, the optimal proxy content for scalable CDNs consists of fewer and less popular files as the total request rate increases and/or as the ratio of the average cost of a proxy stream to the average cost of an origin stream increases.

We found that, if client bandwidth is limited (e.g., if they can listen to only 1.2 simultaneous streams, and thus BWSkim[/U](1.2) is used), as might be the case for transmission of high quality

video over non-broadcast networks, the optimal total delivery cost is significantly higher than the optimal cost if the clients can receive at least two simultaneous streams. For example, for CDNs with ten proxy servers and average cost of a proxy stream that is 10% of the average cost of an origin stream, the increase in delivery cost is up to 200%, for large cache sizes. Furthermore, if the origin server uses multicast delivery, instead of unicast, the total CDN delivery cost is greatly reduced, motivating more effort into a wider deployment of IP or application layer multicast protocols in the Internet.

Chapter 7

Designing Scalable CDNs Using More Precise Network Bandwidth Costs

The previous chapter developed design models to determine the (near) optimal proxy content for scalable streaming CDNs assuming fixed server placement and a restricted server selection and routing strategy where each client contacts either a fixed proxy server or the origin server. Those CDN design models use approximate network bandwidth costs, represented by a constant ratio of the average cost of a proxy stream to the average cost of an origin stream. This chapter develops more elaborate and new CDN design models to determine the server content, server placement, server selection and routing that minimizes the delivery cost function defined in equation (3) in chapter 3, which includes more precise network bandwidth costs that take into account the bandwidth at each hop in the network path traversed from server(s) to client sites.

We use the result from the previous chapter that states that the optimal proxy content includes only full files (instead of file prefixes) for a large region of the design space to assume that each file is *fully* replicated at a number of proxy servers. The design models will be solved to determine the number and placement of file replicas in the network, the selection of the replica that each client requests should be sent to and the routing of streams that the clients receive. Furthermore, the solutions obtained with the models will allow us to assess the cost-effectiveness of adding proxy servers to a CDN using a more accurate estimate of the minimum (server and network) delivery

cost than the one used in chapter 6. Throughout this chapter, we refer to a server, either the origin or a proxy, as a replica server or simply a replica.

Server placement and routing have been previously addressed in a number of studies, mainly in the context of conventional unicast delivery. However, for scalable streaming protocols, the optimal design of a CDN is much more complex as it has to deal with several new tradeoffs introduced by multicast delivery. In particular, as will be shown later in this chapter, the multicast trees that minimize total network bandwidth involve a complex tradeoff between minimizing distance and maximizing the number of clients that share the path segments. Therefore, unlike for unicast delivery, closest server and shortest path routing are not necessarily optimal strategies for scalable delivery. Also unlike for unicast delivery, routing decisions for different clients are not independent in scalable CDNs. Furthermore, computing the cost of a given replica placement involves finding the set of delivery trees from the replicas to the clients that minimize total bandwidth costs. In other words, placement and routing are not independent problems and must be solved jointly. However, the optimal solution for neither of them is known a priori. This greatly complicates the optimal CDN design problem. In section 7.3.1, we provide some quantitative examples to illustrate these complex tradeoffs.

The remainder of this chapter is organized as follows. Sections 7.1 and 7.2 describe our methodology for creating realistic hop level Internet topologies and the specific system assumptions made for developing the CDN design models, respectively. Section 7.3 introduces the new CDN design models and the optimal solutions for a number of example systems. Insights into efficient placement and routing strategies for scalable delivery are derived in section 7.4 and used, in section 7.5, to develop efficient approximate algorithms. Section 7.6 summarizes the chapter.

7.1 Internet Topologies

This section describes our methodology for creating network topologies, which are input parameters for the CDN design problem. The topologies should be realistic representations of the network where the CDN will be built so that the solutions obtained can be directly applied to practice. Here, we focus on creating Internet topologies. Thus, using realistic data that reflects the available connectivity of the Internet is preferred over random topology generators [6, 8, 29, 34, 89, 102,154].

A hop level Internet topology can be created at different levels of abstractions. At a higher lever, it is represented by an Autonomous System (AS) graph. An AS is a collection of routers and links that are under the same administrative domain. An AS is seen to the rest of the Internet as a single routing unit. BGP (Border Gateway Protocol [124]) is the protocol used to exchange path information between ASes and to create routing tables with information on how to reach each other. Router level topologies have a more detailed representation of the network, including individual routers and router interfaces. In either case, the topology can be represented as a weighted graph, where weights represent network capacity or any economic factor that may be associated to each path (e.g., the cost of traversing an AS). For simplicity, we do not include such weights in the examples examined in this chapter. However, they can be very easily added to the topologies and to the methods developed.

For a given set of client sites and access points for replica server placement, a topology might be obtained from the network service providers, from a database such as the CAIDA Project database [32] or from BGP tables and traceroute experiments, such as those described below.

Table 19: Example Client Sites

Symbol	Server Location	Symbol	Server Location	Symbol	Server Location
AZ	Univ. of Arizona, US	CMU	Carnegie Mellon Univ., US	CO-AU	Connect, Australia
CU	Carleton University, Canada	EI-CH	Eimer, Switzerland	GU-UK	Glasgow Univ., U. K.
IN-AU	iiNet, Australia	MO-ES	Metrored Online, Spain	MU-AU	Monash Univ., Australia
OC-ES	OCEA, Spain	OI-IT	Officine Informat., Italy	PW-FR	PacWan, France
SE-FR	Univ. de St. Etienne, France	SDSC	San Diego Super-Computer Center, US	ST-IN	National Centre for Software Technology, India
SU	Stanford Univ., US	TE-AU	Telstra, Australia	TX	Texas A&M Univ., US
UCB	UC-Berkeley, US	UG-CH	Univ. of Geneva, Switzerland	UO	Univ. of Oregon, US
UW	Univ. Wisconsin, US	VY	Vineyard.NET, US	XT-CA	XeniTec, Canada

The exact rules for charging multicast media streams in the Internet are still an open problem. It is not clear whether a stream should be charged for each AS or for each router that it crosses. Nevertheless, the optimal solution for a CDN design problem is determined in the same way for either type of network topology. Therefore, we create both AS level and router level topologies. We create topologies for sets of up to 12 client sites selected from the list of widely distributed client sites, shown in Table 19. These sites include universities, research labs and small Internet Service Providers (ISPs) located in four continents and represent sites where clients for distance education content or for other type of media content, such as entertainment content, might be located.

Although the number of client sites used in our example topologies might seem small, we point out that they are widely dispersed. Intuitively, each client site can approximately represent a total client load that is actually distributed among a set of neighboring nodes. The dispersion of the client nodes, measured, for instance, by the maximum (or the average) distance between any pair of

client sites, is likely to have a stronger impact on the optimal placement and routing than the exact distribution of client load in a number of nodes located in a small neighborhood. Thus, the insights and conclusions obtained for our example client populations should be applicable to systems of similarly dispersion but perhaps with a larger number of client sites.

Each site in Table 19 has a public traceroute server [140], which is useful for obtaining network topology data, as described below. We point out that, unlike many previous studies, our topologies consists of *directed* graphs to account for asymmetric paths, as observed in [97]. The following two sections describe our methodology to create router level and AS level topologies. Appendix D shows a few examples of the topologies created.

7.1.1 Router Level Topologies

The router level topology for a given set of client sites can be created by running a number of traceroute experiments[5], over an extended period, from each client site to every other client site. Traceroute experiments should also be performed from each access point for replica placement to each client site, although many of these traceroute experiments might be omitted without significant loss of topology information if the access points appear in the traceroute data obtained at the client sites. In either case, the number of traceroute experiments should be large and the period during which they are performed should be long enough in order to collect as many different existing routes between any two nodes as possible. For the topologies used in this thesis, over 1000 traceroutes were executed from each client site to every other client site, over a period of four weeks in May 2002. On average, at least 150 traceroutes between a source and a destination were

[5] Each traceroute returns the sequence of routers (or router interfaces) reached along the path from one client to the other.

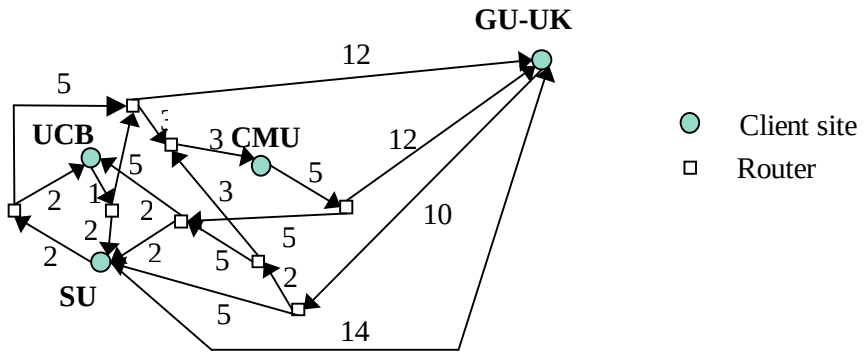


Figure 42: Example Router Level Topology

successful. A traceroute is not considered successful if it does not return a complete path from the source to the destination.

Figure 42 shows an example of a router level topology for four client sites selected from Table 19, obtained using the traceroute experiments described above. We do not consider the bandwidth at the routers located at the same domain name as the client sites as part of the total network delivery cost. Thus, in the topologies, each client site is connected to the first router in the traceroute path that is not in the same domain name. However, those routers can be easily added to the topology if necessary. Furthermore, to simplify the representation of the topology as well as to reduce the solution time of our techniques, we include in the topologies only the nodes that are located at path intersections, and label each path segment with a "weight" equal to the number of routers in the path. Parallel paths that are longer than the given paths and do not intersect with any other path are not included in the topology because they will never be selected as part of the optimal routing. The resulting condensed topologies are used in the remainder of this chapter.

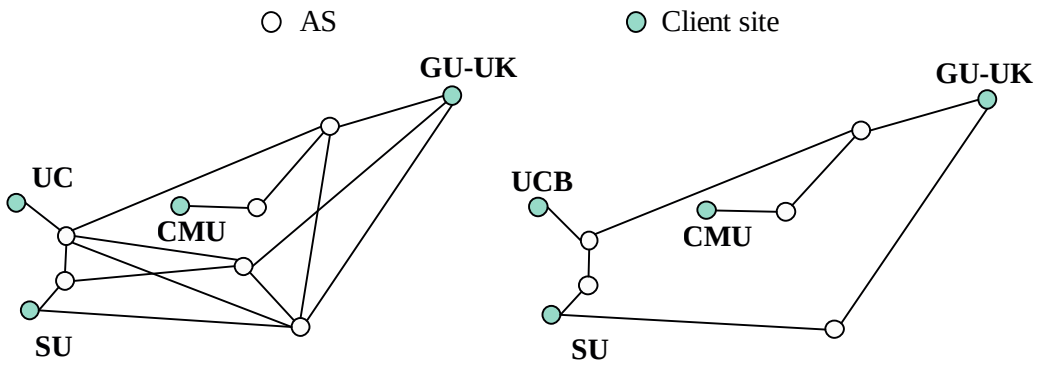
7.1.2 Autonomous System Level Topologies

The main sources of information for creating AS-level network topologies are publicly available BGP routing tables. Each BGP routing table, collected at a source AS, contains routing information

from the source to different destination ASes. Each entry in a BGP routing table specifies an AS path, AS_1, AS_2, …, AS_n, from the source located in AS_1 to a destination address prefix located in AS_n. One publicly available source of BGP routing tables is the University of Oregon Route Views project [126]. A collector (the Route Views router) connects to (or peers to) and collects routing information from dozens of different ASes around the world. The RIPE (Réseaux IP Européens) runs a similar project called RIS (Routing Information Service). It has seven collection points that peer to different ASes [125].

There are at least two previous approaches for creating AS level network topologies for a given set of sites: a) using the AS connectivity data from BGP routing tables (as done in [115]) or b) using the router level topology and mapping each router IP address to the AS it belongs to (as done in [85, 131]). The connectivity available at the BGP tables represents only contractual relationships between ASes rather than whether traffic is actually routed between them [26]. Furthermore, BGP tables show only the selected (best) routes rather than all possible routes stored in the router. Thus, using only BGP tables to obtain a topology may incur significant distortion of network connectivity [26]. Therefore, the latter method provides a much more accurate representation, assuming the router level topology includes all possible replica access points. In order to map an IP address into the AS it belongs to, one approach, used by the Skitter project [131] to create an AS level map of the Internet, is to use the mapping of IP address prefix to AS available at the BGP tables.

We used BGP tables collected by both RouteViews and RIPE to create AS topologies using both methods. The topologies created by each method for the same four client sites used in Figure 42 are shown in Figure 43. The topology in Figure 43(a) is created using only the BGP tables. To simplify the figure, we show only the paths between each client pair of up to 3 hops in length. The



a) BGP tables only (paths of 3 or fewer hops) **b) BGP tables and traceroute data**

Figure 43: AS Level Topologies for Example Set of Client Sites (all links are bi-directional)

topology in Figure 43(b) was created from Figure 42 and the mapping of each router IP address to the AS it belongs to, using the information in the BGP tables (as suggested in [131]). Note that Figure 43(a) contains paths that are not observed in any of the many traceroutes we ran. Thus, the topology in Figure 43(b) is more reliable. AS topologies used in the remainder of the chapter are created by mapping from the router level topology.

7.2 System Assumptions

The main system assumptions made throughout this thesis for developing our CDN design methods were discussed in chapter 3. This section elaborates on a few extra assumptions specific for the methods developed in this chapter.

The multicast delivery tree for each server is assumed to have a fixed topology that is optimized for the given client sites (and corresponding current client request rates) and set of optimal replica placements, where a client site is a group of clients located in the same domain name (for router topologies) or in the same AS (for AS topologies). In other words, the topology does not change every time a client joins or leaves the multicast group. For popular objects, clients

from different sites are continually joining the multicast tree. Thus, we expect a fixed topology optimized in this way to have high performance.

However, we point out that, as with any CDN, including web CDNs, the delivery trees and, possibly, the replica placement might need to be updated in response to significant changes in the workload. In particular, if the client workload changes significantly from day to day, for instance, the CDN designer must decide whether it is more cost-effective to update the CDN configuration after each day (e.g., during the night) to reflect the expected load on the following day or whether it is more cost-effective to keep the same CDN configuration (i.e., the one that is optimized for the day with highest total load) during multiple days. In that case, the CDN designer must compare the cost of updating the CDN configuration after each day and the cost increase (compared to the optimal cost) if the same CDN configuration is used during multiple days. We also point out that the number and placement of servers may need to be updated only when the set of client sites change. If the client sites remain the same but the request rates from each client site change, the optimal CDN design can be constructed by simply updating the content stored at each server, which is a much cheaper operation than moving servers around the network.

The total network bandwidth is expressed as a weighted sum of the bandwidth required for each hop in the delivery tree, where a hop is an AS in AS level topologies, or a link between two routers, in router level topologies. Per hop network bandwidth, expressed as the average number of concurrent streams sustained through a hop, as well as server bandwidth are calculated, as in chapter 6, using the previously derived formulas for required bandwidth for scalable delivery protocols. The cost models developed in chapter 6 assumed the bandwidth skimming protocol and calculated server bandwidth using equation (1) (in section 2.5.2). The cost models developed in this chapter are general enough to be applied to different delivery protocols such as patching [82] and

periodic broadcast [83], in addition to bandwidth skimming. Given N, the request rate (or demand), of the clients that share a given server (or a given hop of the delivery tree), the average number of concurrent streams from the server or over a network hop are calculated using equation (1) for bandwidth skimming. Similarly, server or per hop network bandwidth is given by $B(N) = \sqrt{2N+1} - 1$, for patching [62] and by a constant, say B = 8, for periodic broadcast, where the constant is determined based on the desired client startup latency for the server and for each segment of the delivery tree.

In this chapter, we develop methods for constructing the minimum cost CDN for a *single* file, recognizing that each file may have a different optimal number and placement of servers. Furthermore, we assume that the file is *fully* replicated at all servers based on our result from the previous chapter that show that full file caching outperforms prefix caching in a very large region of the design space where replication of a popular file is desirable. The methods can be modified in the future to account for client interactivity by using the appropriate bandwidth calculations such as those recently proposed in [88, 135]. Furthermore, the solution for multiple files sharing the same placement can be easily formulated from the single file case. The methods developed in this chapter can also be applied to multiple files by solving them separately for each category of file, where a category is defined by "similar" client request rates, and then defining a method for merging the solutions in order to obtain a global solution with (near) optimal delivery cost. Other future extensions to our methods, discussed in chapter 8, include the *simultaneous* solution for multiple files allowing varying constraints on sharing of placement and routing among the files.

Finally, we consider optimal and near-optimal multicast delivery trees that may not be suitable to be implemented using current IP multicast protocols, which, in theory, construct reverse shortest path trees [63, 81]. However, the delivery trees may be constructed in application layer multicast

[45]. Sections 7.4 and 7.5 compare the delivery costs of the solutions with the (near) optimal trees against solutions where each delivery tree is the tree of shortest paths.

7.3 Optimal Replica Placement and Routing

This section introduces the new design models that are used to determine the file replica placement and routing that minimize total delivery cost of a scalable CDN. Section 7.3.1 provides some quantitative motivating examples that show that the CDN that has minimum total delivery cost for conventional unicast streaming of a given file may be significantly suboptimal if a scalable multicast delivery protocol is used. Sections 7.3.2 and 7.3.3 provide the simple basic formulation of the minimum cost model for scalable CDN as well as modifications to the model that are required to produce exact solutions. Section 7.3.4 provides some optimal systems computed from the model for a number of example Internet topologies and client loads.

7.3.1 Motivating Examples

This section provides some quantitative examples to illustrate that: (a) the replica placement and routing strategies that minimize total delivery cost for a CDN that uses conventional unicast streaming may be very different from the optimal strategies for a scalable streaming CDN, (b) if the strategies that are optimized for conventional streaming are used for scalable delivery, the increase in total delivery cost may be significant and, thus, (c) designing optimal scalable systems is a complex optimization problem.

In Figures 44 and 45, we consider two network topologies that capture the router level topology of the Internet 2 backbone [56] with client sites located close to a number of nodes in the backbone. Client sites are indicated by circles, darkly shaded when they are locations for replicas

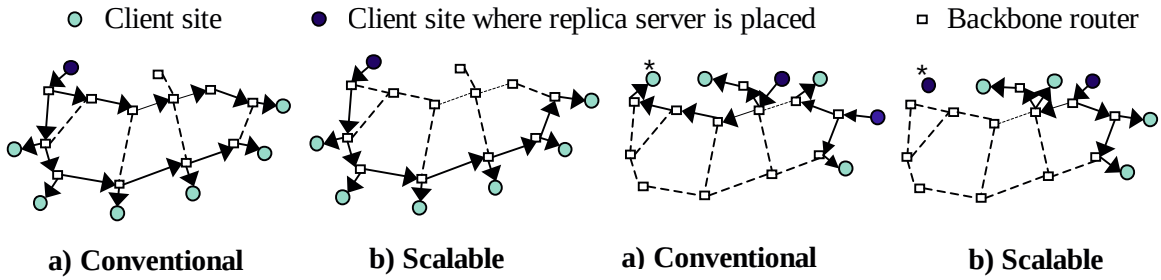


Figure 44: Optimal Routing (fixed replica placement)

Figure 45: Optimal Replica Placement and Routing

and lightly shaded otherwise. Network paths that are not used by any delivery tree are shown as dashed lines and the optimal routing (delivery trees) are shown as directed arcs.

Figure 44 shows results for optimal routing to six client sites given a fixed replica placement at a seventh client site. Figure 44(a) shows the optimal routing for conventional unicast streaming, while Figure 44(b) indicates the routing for scalable streaming. In both cases, all clients are assumed to have equal demand for the objects stored at the server. The optimal routing in Figure 44(b) includes a path to the rightmost client site in the diagram that is not the shortest path. The reason for this is that the average bandwidth requirement for the first five hops in the shortest path is higher than the incremental bandwidth along the six first hops of the optimal path, which is shared among other client sites. This example illustrates that the optimal routing for a client does depend on the routing decisions made for other clients. If the CDN in Figure 44(a) is used for scalable multicast streaming instead of that in Figure 44(b) and if the request rate from each client site is $N = 1000$, there is a 26% higher cost in terms of network bandwidth.

Figure 45 shows the optimal placement of two replicas and the optimal routing for six client sites, assuming all clients have equal request rate N except for the client marked with '*', which has request rate equal to $N/2$. For conventional unicast streaming, Figure 45(a) shows that the replicas are optimally placed close to the client sites that have higher request rate (i.e., the higher

demand client sites), while the lower demand client site is served through a four-hop path. In the optimal scalable system, shown in Figure 45(b), one replica is placed at the lower demand client site. The average distance from the higher demand client sites to its closest server is slightly higher than in the conventional system, but so is the degree of sharing in those paths. The bandwidth savings from avoiding the four-hop path to the lower demand client site offsets the increase in bandwidth in the paths to the high demand client sites. For N = 1000, if the system in Figure 45(a) is used for scalable delivery instead of the system in Figure 45(b), total network bandwidth increases by 26%.

Furthermore, unlike for unicast CDNs, where the total server bandwidth does not depend on the number of replicas, the total server bandwidth for scalable systems increases significantly as the number of replicas increase, because each server has bandwidth that is sub-linear in its client request rate. As an example, for the system shown in Figure 45(b), the total server bandwidth increases by 40% if a third replica is (optimally) added to the CDN.

7.3.2 Basic Optimization Model

This section develops the basic formulation of the CDN design model that determines the number and placement of replicas and the delivery tree(s) that minimize total delivery cost, expressed as a (weighted) sum of the total network delivery cost and the total server delivery cost, as shown in equation (3) (chapter 3). The optimization model is solved to determine the optimal placement and routing (i.e., multicast delivery trees) for a given number of replicas and for given network topology, client sites and request rates, locations of the access points for replica placement, and the relative cost of per hop network bandwidth and server bandwidth. The optimal number of replicas is determined by comparing the minimum delivery costs for different number of replicas and choosing the most cost-effective option.

Table 20: Input Parameters and Outputs of the Cost Model for Placement and Routing

Parameter	Description
V	Set of nodes {1, 2, 3, … n}
A	Set of *directed* arcs $\{(i,j)\}$, $i, j \in V$
$w_{i,j}$	Weight on arc $(i,j) \in A$ (i.e., relative cost of one unit of bandwidth on the arc)
S	Access points for replica (i.e., server) placement $(S \subseteq V)$
C_V	Client sites $(C_V \subseteq V)$
N	Set of client loads: $N = \{N_i: N_i > 0$ if $i \in C$ and $N_i = 0$ if $i \in V - C\}$
m	Number of replicas to be placed
γ	Ratio of cost of one unit of server bandwidth to cost of one unit of network bandwidth over a hop
S^*	Set of replica server nodes in the optimal delivery network $(S^* \subseteq S)$
$N_{i,j}$	If arc (i,j) is in a multicast tree, sum of loads of the clients that are served via arc (i,j)
$B_{network}$	Total network bandwidth summed over all arcs
B_{server}	Total concurrent server bandwidth summed over all servers

The subset of all parameters, defined in chapter 3, that are used for defining the CDN design models presented in this chapter are summarized in Table 20. They include the set of nodes, arcs and arc weights that define the network topology, the set of access points for replica placement, the client sites and their demands (request rates or loads), the number of replicas to be placed and the ratio of the cost of one unit of server bandwidth to the cost of one unit of per hop network bandwidth. The main outputs of the model, also shown in Table 20, are the optimal replica placement (the set of nodes S^*), and the optimal routing (i.e., multicast delivery trees), which is determined from the loads $N_{i,j}$, for all arcs (i,j) in the set A. The load on an arc is defined as the sum of the request rates of all clients that share that arc in the delivery tree. The model also computes total network bandwidth and total server bandwidth as a function of these variables. We assume all hops have equal cost per unit of bandwidth and the weights are used to represent the number of hops in each path segment of the condensed topologies created as described in section 7.2.

minimize $B_{network} + \gamma B_{server}$
$S^*, N_{i,j}$

subject to: (1) $|S^*| = m$

(2) *for all* $j \notin S^*$: $\sum_{(i,j)\in A} N_{i,j} = N_j + \sum_{j,k)\in A} N_{j,k}$

where:

Bandwidth Skimming: $B_{server} = \sum_{i\in S^*} \log(\sum_{(i,j)\in A} N_{i,j} + 1)$ and

$$B_{network} = \sum_{(i,j)\in A} w_{i,j} \log(N_{i,j} + 1) \quad \forall (i,j) \in A$$

Patching: $B_{server} = \sum_{i\in S^*} (\sqrt{2 \sum_{(i,j)\in A} N_{i,j} + 1} - 1)$ and $B_{network} = \sum_{(i,j)\in A} w_{i,j} (\sqrt{2N_{i,j} + 1} - 1) \quad \forall (i,j) \in A$

Periodic Broadcast: $B_{server} = m$ and

$$B_{network} = \sum_{(i,j)\in A} w_{i,j} \vartheta_{i,j} \quad \textit{where} \quad \vartheta_{i,j} = \begin{matrix} 1 & \textit{if} \quad N_{i,j} > 0 \\ 0 & \textit{otherwise} \end{matrix} \quad \forall (i,j) \in A$$

Conventional (unicast): $B_{server} = \sum_{i\in C} N_i$ and $B_{network} = \sum_{(i,j)\in A} w_{i,j} N_{i,j} \quad \forall (i,j) \in A$

Figure 46: Optimization Models for Defining Replica Placement and Routing for Scalable CDNs

The fundamental optimization model is defined in Figure 46. The goal is to minimize the total delivery cost over the possible values of S^* and $N_{i,j}$, for all arcs (i,j) in set A. The optimization is subject to constraints that guarantee that: (1) the number of replicas is equal to the input parameter and (2) total traffic out of any node that is not chosen for placement of a replica is equal to the total flow into the node minus the load at the node (which is nonzero if the node is a client site). Note that the model described in Figure 46 does not include constraints to reflect the capacity (i.e., maximum bandwidth) of a given network hop or the maximum bandwidth that any given replica access point can sustain. However, we point out that these constraints could be very easily added if

necessary. We also point out that the model can be used to compute the optimal routing for a given replica placement by specifying S^* as part of the model input and solving only for the loads $N_{i,j}$.

The model can be applied to any of the previously proposed scalable streaming protocols by using the appropriate formula to compute B_{server} and $B_{network}$. The formulas for bandwidth skimming, patching and periodic broadcast are shown in Figure 46. The bandwidth calculations for conventional unicast streaming are also provided, for the sake of completeness. Recall that the required server bandwidth for bandwidth skimming depends on the available client receive bandwidth, which determines the value of the constant η. Instead of choosing a specific client bandwidth, we choose to follow the same approach as in [155] and use $\eta = 1$, derived in [61] for infinite client bandwidth. In that case, the formula is a lower bound on the required server and network bandwidth for any client bandwidth. We expect the main insights and conclusions to be the same for different values of η, i.e., different client bandwidth.

For a replica server connected to a given AS, in an AS level topology, the network delivery cost may include the bandwidth inside the AS. In this case, the replica access point should be an extra node (i.e., a "stub") that is connected only to the AS node and has no client load. In other words, in an AS topology, replicas are placed at stub nodes. A link from a stub to AS A represents the utilized links within A; a link from AS A to AS B represents the utilized links within B and so forth. Similarly, for router level topologies, for a replica server that is connected to an internal router, a stub is used to represent the bandwidth to route from the network where the server is actually placed to that router. This is because an internal router may be actually a transit router that is not part of the network where the replica is placed.

We have presented definitions for B_{server} and $B_{network}$ for four protocols. For unicast streaming and periodic broadcast, the model is a mixed integer linear programming problem and can be

efficiently solved. For bandwidth skimming and patching, the model includes logarithm or square root functions, which are non-linear and non-convex. As in the previous chapter, for such models, the optimization algorithms do not guarantee true optimality. Next section describes the techniques used, one of which is similar to the one used in the previous chapter, in order to guarantee true optimality in the model solution.

7.3.3 Linear Optimization Model

This section describes the modifications applied to the bandwidth skimming model in order to guarantee true global optimality. Similar modifications can be made to the patching model. It also introduces new constraints that are based on our knowledge of the optimal configuration and are added to the model in order to improve the solution time.

We use two techniques for replacing the logarithmic functions by expressions that are modeled with linear functions and binary variables. In either case, the modified model is a mixed integer programming problem and can be solved to global optimality. One technique is similar to the one described in chapter 6. We use a table of pre-calculated bandwidth with one entry for every possible values of $N_{i,j}$ (partial sums of the client loads) and use binary variables $\theta_{i,j,k}$ and $\delta_{i,k}$ to indicate whether the load $N_{i,j}$ on a link (i,j) or the total load for a server, respectively, is equal to entry k in the table. B_{server} and $B_{network}$ are then redefined as linear functions of $\theta_{i,j,k}$ and $\delta_{i,k}$ and the pre-computed bandwidths.

The solution time for the model that uses the table of pre-computed bandwidths becomes prohibitively long as the number of partial sums increases. To improve the efficiency of the algorithm, we use an iterative strategy based on the solution of increasingly accurate models. We start by constructing a table with fewer entries. The solution obtained with the model with a shorter

table is not necessarily optimal but can be used as a starting point for a more accurate model that uses more entries in the table. This process is repeated until the model with the complete table is solved.

The second technique approximates the logarithm function by a piecewise linear function. Because of the asymptotic nature of the logarithm function, a good approximation can be obtained with a small number of pieces. Binary variables are introduced to indicate in which "piece" the load $N_{i,j}$ lies. As with the first technique, the number of binary variables (equal to the number of pieces times the number of arcs) may be very large, resulting in very long solution times. As before, an iterative strategy can be used, where the solution of a model with fewer pieces is used as a starting point for a more accurate model.

Further improvements in the solution time are achieved by adding the following extra constraints to the model, which reduce the search space that must be examined by the optimization algorithm:

- A client node has only one parent in the delivery tree, i.e., for each node $j \in C_V$, there is only one arc $(i,j) \in A$ with nonzero $N_{i,j}$.
- A node that is neither a client nor a server has at most one parent in the delivery tree; that is, for each j with $j \notin C_V$, $j \notin S^*$, $N_{i,j} \neq 0$ for at most one arc $(i,j) \in A$. Furthermore, such nodes may have non-zero load on out-flowing arcs only if they have nonzero load along one inflowing arc.
- Since we assume each replica is the full file and there are no constraints on server resources, each server is the root of a delivery tree and thus has zero load on all inflowing arcs. That is, $N_{i,j} = 0$ for all $j \in S^*$ and all $(i,j) \in A$..

These modifications reduce the model solution time significantly allowing it to be applied to realistic networks with up to 12 client sites with heterogeneous loads, interconnected by 100 nodes that represent path intersections, another 100 nodes that represent possible replica access points and replica placement constrained only to the client sites and to the access points. Larger problems might also be tractable if the heuristic solution, developed in section 7.5, is used as the starting point for the model to determine the optimal solution. Experimenting with this hybrid strategy is left for future research.

7.3.4 Results for Example Client Populations

In this section, the optimization models are applied to a set of realistic Internet topologies with example client populations in order to evaluate the solution feasibility and quantify the total network and server bandwidth cost increase if the replica placement and routing that are optimal for conventional unicast streaming are applied to scalable multicast systems.

The overall approach we take is to solve the optimization model for different sets of client sites and, for each case, compare the optimal delivery cost for the scalable CDN against the cost of using scalable delivery in a CDN that is designed, for the same set of client sites, using the placement and routing that are optimized for conventional unicast delivery. The optimal placement and routing for unicast delivery are computed using the unicast model. Given a solution obtained with this model, we calculate the total cost if multicast streaming is used within it. In general, this cost comparison could be done for any of the scalable delivery protocols. Given its efficiency, we use the bandwidth skimming protocol for the experiments reported in this chapter.

Since we do not have a quantitative relative cost of server and per-hop network bandwidth, we set $\gamma = 0$. In this case, the goal of the optimization model is to minimize the total network

bandwidth, but it also reports the total server bandwidth for the solution. We plot the total server bandwidth (B_{server}) and the total network bandwidth ($B_{network}$) as a function of the number of replicas, so that the main factors that determine the total delivery cost are quantified for each solution.

We have applied this methodology for a number of different AS and router level topologies with client populations with different degrees of dispersion. For example, we have solved for cases where all clients are located in different regions of the same country (e.g., the United States), in North America, in Europe and North America and in four different continents (America, Europe, Asia and Australia). Furthermore, we also varied the heterogeneity in the client's request rates from no heterogeneity (i.e., homogeneous client request rates) up to 25-50% of the clients having lower request rates than the others. The precise values of the request rates are not as important as their magnitude. Thus, we experiment with client request rates equal to one of three values: N_i = 10, N_i = 100 or N_i = 1000.

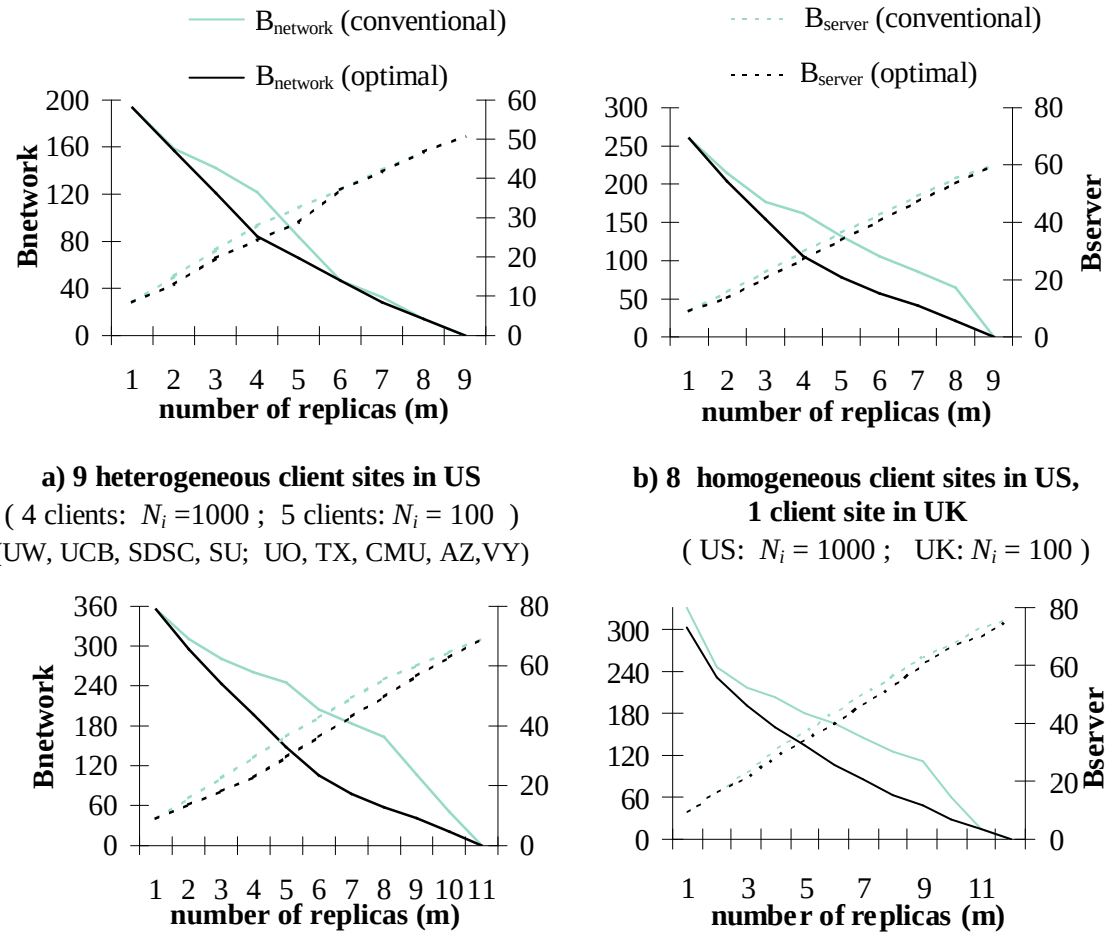
The sets of client sites, represented by their symbols in Table 19, and the corresponding heterogeneous demands, used in the topologies considered in this chapter are shown in Table 21. A '–' indicates that a particular client site is not included in the set. Table 21 also shows, for each set, the number of client sites, the number of continents covered by the client sites and the maximum distance (in number of hops in the router level topologies) between any pair of client sites in the set, as measures of client dispersion. For each set, we have created AS and router level topologies with heterogeneous client demands (as shown in the table) and with homogeneous client demands. Figures 47, 48 and 49 show a few representative results of these experiments. The caption of each figure shows the client sites, identified by the corresponding symbol in Table 19, client demands and the type of topology (AS or router level) used.

Table 21: Sets of Client Sites Used to Analyze Alternative CDN Solutions

Client	**Heterogeneous Client Demands N_i (/ 1000)**						
	Set 1	Set 2	Set 3	Set 4	Set 5	Set 6	Set 7
AZ	0.1	0.1	1	1	1	1	1
CMU	0.1	0.1	0.1	1	1	1	1
CU	–	–	–	–	0.1	–	1
GU-UK	–	–	–	0.1	0.1	0.1	0.1
MU-AU	–	–	–	–	–	0.1	–
SDSC	1	1	1	1	1	1	1
ST-IN	–	–	–	–	0.1	1	–
SU	1	1	1	1	1	1	1
TE-AU	–	–	–	–	–	0.1	–
TX	0.1	0.01	1	1	1	1	1
UCB	1	1	1	1	1	1	1
UO	0.1	0.1	0.1	1	1	1	1
UW	1	1	1	1	1	1	1
VY	0.1	0.01	1	–	–	–	–
UG-CH	–	–	–	–	–	–	0.1
SE-FR	–	–	–	–	–	–	0.1
Number of Client Sites	9	9	9	9	11	12	12
Number of continents	1	1	1	2	3	4	2
Max distance between client sites (in router hops)	17	17	17	19	22	22	20

Figure 47 shows results for four router level topologies with heterogeneous client sites with different degrees of dispersion. For each topology, the graph shows the total network bandwidth (solid curves) and the total server bandwidth (dotted curves) as a function of the number of replicas. These results illustrate that using placement and routing strategies that are optimized for conventional delivery to design CDNs that use scalable delivery may result in significant network cost increase, compared to the optimal (50-150% for large number of replicas).

Figure 48 shows the results for two AS topologies with heterogeneous clients. The observed network cost increases are much lower (i.e., up to 27%). Note that the sets of client sites used in the



a) 9 heterogeneous client sites in US
(4 clients: N_i =1000 ; 5 clients: N_i = 100)
(UW, UCB, SDSC, SU; UO, TX, CMU, AZ,VY)

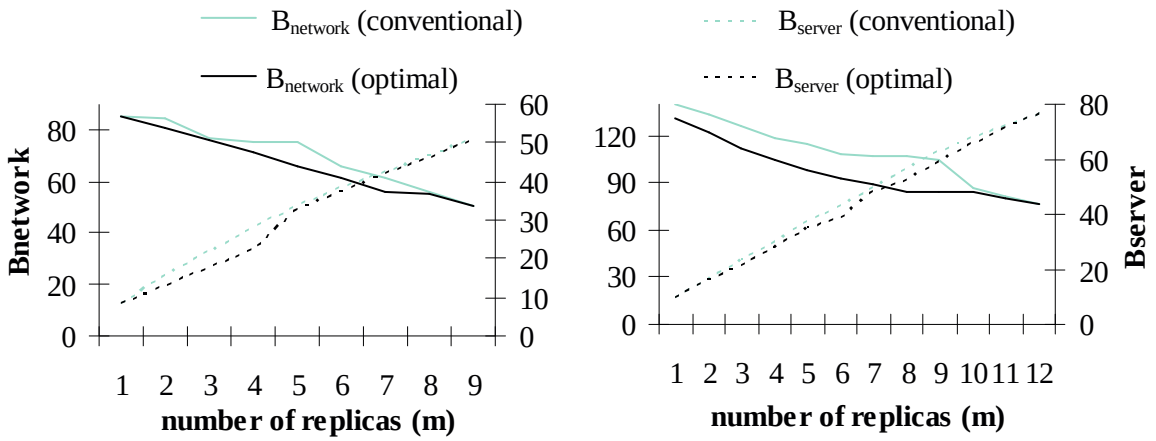
b) 8 homogeneous client sites in US, 1 client site in UK
(US: N_i = 1000 ; UK: N_i = 100)

c) 8 client sites in US, 1 in Canada, 1 in UK, 1 in India
(US: N_i = 1000 ; UK, Canada, India: N_i = 100)
(AZ, CMU, SDSC, SU, UCB, UW, TX, UO; CU, GU-UK, ST-IN)

d) 8 client sites in US, 1 in India, 1 in UK, 2 in Australia
(US, India: N_i = 1000; UK, Australia: N_i = 100)
(AZ, CMU, SDSC, SU, UCB, UW, TX, UO, ST-IN; GU-UK, TE-AU, MU-AU)

Figure 47: Cost Comparison of Optimal Conventional and Scalable Delivery Systems for Router Level Topologies (heterogeneous clients)

AS topologies for Figures 48(a) and 48(b) are the same as those used for the router topologies in Figures 47(a) and 47(d). The reason for the different cost increase for router and AS level topologies is the different lengths of the paths between client sites in the topologies and will be clear in the next section. The longer the lengths of the paths, the higher the cost increase associated



a) 9 heterogeneous client sites in US

(4 client sites: N_i =1000 ; 5 client sites : N_i =100)

(UW, UCB, SDSC, SU ; UO, TX, CMU, AZ, VY)

b) 8 client sites in US, 1 in India, 1 in UK, 2 in Australia

(US, India: N_i = 1000, UK, Australia: N_i = 100)

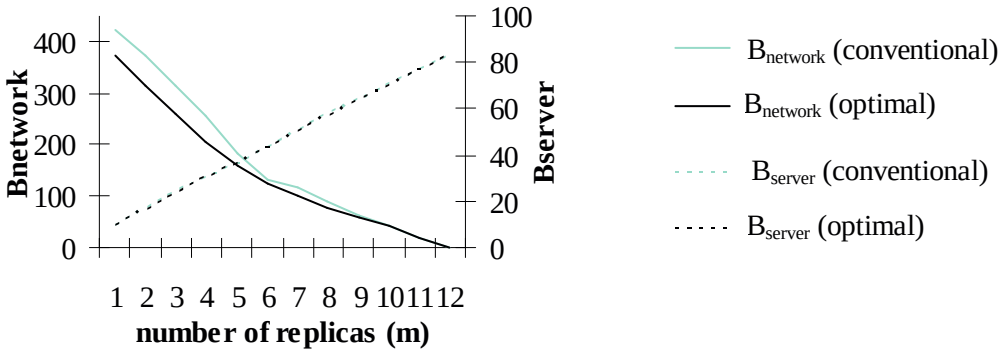
(AZ,CMU,SDSC, SU,UCB,UW,TX,UO, ST-IN; GU-UK, TE-AU, MU-AU)

Figure 48: Cost Comparison of Optimal Conventional and Scalable Delivery Systems for AS Level Topologies (heterogeneous clients)

with the use, for scalable streaming, of placement and routing optimized for conventional unicast streaming can be.

Figure 49 shows that the cost increase is also much less significant for homogeneous client sites. The maximum cost increase observed for all systems considered was only 15%. However, next section will show that the potential cost increase from using strategies optimized for conventional unicast delivery to design scalable delivery systems is significant, even if the client sites have homogeneous loads.

The results in Figures 47-49 also show that the total server bandwidth for scalable CDNs designed using placement and routing optimized for conventional delivery may be significantly higher than the optimal (i.e., up to 30%). Moreover, total server bandwidth increases significantly with the number of replicas because, on average, fewer clients share each server multicast stream. Whereas the benefit from adding replicas in a conventional CDN is clear, Figures 47-49 show that

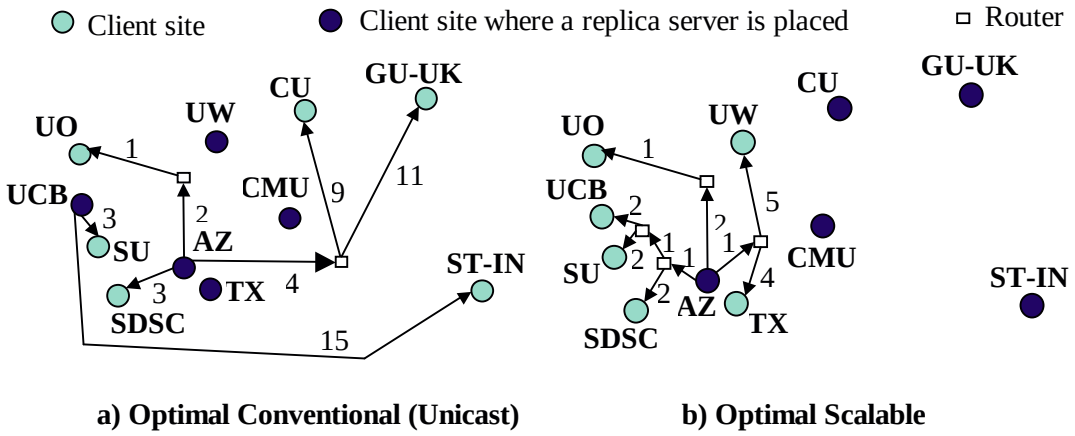


(12 homogeneous client sites: 8 in US, 1 in Canada, 3 in Europe: N_i =1000)
(CMU, SDSC, SU, UCB, UW, TX, UO, CU, GU-UK , SE-FR , UC-GH)

Figure 49: Cost Comparison of Optimal Conventional and Scalable Delivery Systems for Homogeneous Client Sites (router level topology)

adding replicas in a scalable CDN may not be cost-effective. The optimal number of replicas for a CDN depends on the ratio of the cost of server bandwidth, to the cost of network bandwidth. Note that the same conclusion was reached in chapter 6. However, the models developed in this chapter use more accurate network bandwidth costs and, thus, are a more reliable and complete tool for determining whether adding an extra server to a scalable CDN is cost-effective.

Figure 50 shows the optimal replica placement and routing for conventional and scalable delivery for five replica servers in the topology sed in Figure 47(c). The optimal solution for conventional delivery (Figure 50(a)) consists of placing all replicas at the clients with highest demands, which results in delivery trees with long paths to the low demand clients. The optimal solution for scalable delivery (Figure 50(b)) has replicas placed at clients that are farther away (in United Kingdom, Canada and India), even though those clients have lower demand. Thus, some of the high demand clients are served through slightly longer but mostly shared paths. If the conventional design in Figure 50(a) is used for scalable delivery, the total network cost is 66% higher than the optimal in Figure 50(b). Insights into when and why the optimal solutions for scalable and conventional delivery may differ are derived in section 7.4.



a) Optimal Conventional (Unicast) **b) Optimal Scalable**

Figure 50: Alternative Designs for an Example Distribution System
(11 client sites, m = 5 replicas, $N_{CU} = N_{GU\text{-}UK} = N_{ST\text{-}IN} = 100$, $N_{others} = 1000$, router level topology)

Summarizing, the key conclusions from these experiments are: (1) the optimal replica placement and routing for conventional unicast delivery may be significantly sub-optimal for scalable (multicast) delivery, (2) efficient heuristics are needed to obtain approximately optimal strategies for larger number of client sites and (3) the optimal number of replicas (servers) involves a tradeoff between total network bandwidth cost and total server bandwidth cost that is challenging to quantify.

7.4 Insights for Replica Placement and Routing

This section analyzes simple topologies in search for insights into efficient placement and routing strategies for scalable delivery. The quantitative insights are drawn for the bandwidth skimming protocol though we expect them to hold for any protocol that has bandwidth requirements sub-linear (but not constant) with client load. For periodic broadcast or live streaming, in which the network and server bandwidth are independent of the client demand, Fei *et al* developed a number of routing heuristics in [67]. Sections 7.4.1 nd 7.4.2 analyze routing and simple placement strategies, respectively. This analysis leads to the development, in section 7.5, of efficient heuristics for designing scalable CDNs with near optimal delivery cost. Section 7.4.3 provides

insights into why and when the optimal placement for conventional delivery is significantly sub-optimal, as observed in the previous sections.

7.4.1 Insights for Routing Strategies

We are aware of only one previous study that addresses routing for scalable delivery assuming per hop network bandwidth that is dependent on the load at the clients that share the hop [155]. In this work, the authors propose two heuristics that build a multicast tree from a single server that uses bandwidth skimming delivery. The two heuristics build the tree by adding one client at a time, choosing at each step the client that can be added with a) the fewest number of extra links or b) the lowest incremental bandwidth cost. By comparing the two heuristics and the simpler shortest path routing, on large (600 – 6000 node) randomly generated network topologies, the authors find that adding clients based on the incremental cost performs best, but the bandwidth savings compared to shortest path routing are modest (only up to 16%). These results seem to indicate that routing based only on the shortest distance from server to the client sites might be a good heuristic, given that it is very efficient [19] and it does not perform much worse than the more sophisticated and complex heuristic that takes into account both distance and sharing to measure the incremental bandwidth cost associated with a routing decision. Furthermore, compared to shortest path routing, adding clients based on the number of links, which is simpler than the other heuristic, though sometimes better, can result in total cost up to 40% higher.

In this section, we re-evaluate the use of shortest path for scalable routing but, unlike in [155], we use the optimal strategy as benchmark to determine how far from the optimal (in terms of delivery cost) it can be. In other words, we search for the *worst case* performance of the tree of shortest paths. We expect to obtain more conclusive results into the performance of shortest path routing for scalable delivery. We also look for the worst case performance of the tree of fewest

links, if used for scalable delivery. Unlike in [155], we use the actual tree of fewest links instead of a heuristic to build it.

As a starting point, we note that the optimal strategy for protocols such as bandwidth skimming or patching, for which the per hop network bandwidth depends on the load on the hop, lies in the solution of a complex tradeoff between minimizing distance from replica to clients and maximizing path sharing. When choosing between paths of equal length for routing to a new client, it is always more cost-effective to choose the path that allows for more sharing because the incremental bandwidth along that path, due to the new client, is the lowest. When choosing between paths with equal amount of sharing, the shortest path is always the best. Selecting the optimal path becomes significantly more complex if the alternative paths have different lengths and different degrees of sharing. Furthermore, note that the tree with fewest links from a server to a given set of clients allows for the maximum sharing. Thus, each one of the tree of shortest paths and the tree of fewest links represents one extreme of the tradeoff involved in the optimal tree. Furthermore, both the tree of shortest paths [19] and the heuristic to build the tree of fewest links, proposed in [155], are easy to compute. By evaluating the two trees, we evaluate the impact of each of these extremes on the optimal solution, which might lead us to insights into whether more complex strategies, such as the minimal incremental cost heuristic proposed in [155], are more appropriate than these simpler heuristics.

More precisely, this section compares the network delivery costs of two canonical delivery trees, namely the tree of shortest paths (t_{sp}) and the tree with fewest links (t_{fl}) against the optimal, referred to as the network delivery cost of the tree of minimum cost (t_{min}). Let B_{fl}, B_{sp} and B_{min} denote the network delivery cost of each tree, respectively. Using a simple topology, with only two

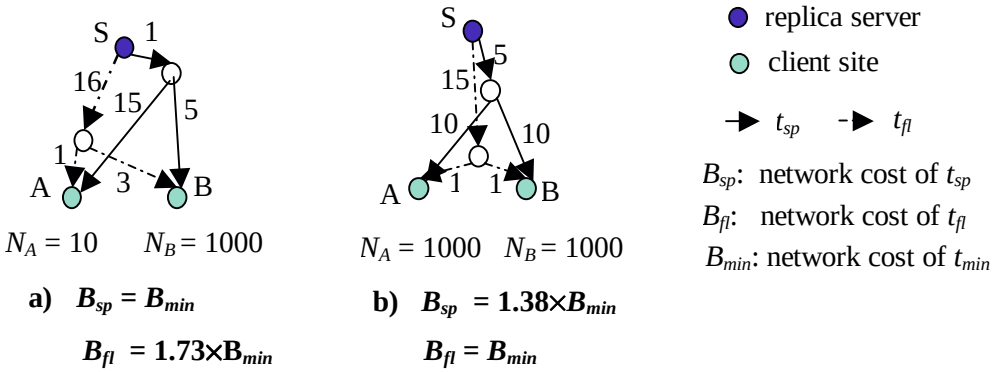


Figure 51: Comparing Tree of Shortest Paths and Tree of Fewest Links Against Optimal

clients, we derive the maximum delivery cost increase, compared to the minimum cost, if each tree is used for scalable delivery.

Before introducing the maximum cost increase associated with t_{sp} and t_{fl}, we show, in Figure 51, two numerical examples that illustrate how each tree can relate to t_{min} and the tradeoff that defines the optimal solution. Each topology in Figure 51 has one server that uses bandwidth skimming to deliver streams to two clients *A* and *B*, with demands N_A and N_B shown in the figure. In each case, the possible trees that can be built, given a hypothetical underlying topology, from the server to the clients are shown. In Figure 51(a), there are two trees. Each path segment in a tree is labeled with the length (in number of hops) of the segment. Assuming the network bandwidth formulas for bandwidth skimming given in Figure 46, it is easy to show that, in Figure 51(a), t_{fl} incurs a 73% higher total network bandwidth for serving clients *A* and *B* than t_{sp}, which is optimal since there are no other paths in this topology. Similarly, in Figure 51(b), there are two trees: t_{fl} is optimal, and t_{sp} uses 38% more network bandwidth. These simple examples show that either tree can be optimal but either one can also incur significant cost increase. Other examples can be easily constructed to show that there are cases where neither tree is optimal.

The examples in Figure 51 also show that the optimal tradeoff between maximizing sharing and minimizing distance is complex. Using shared paths, as in the tree of fewest links, is not cost-effective for scalable delivery if the paths are "much longer" than the (less shared) shortest paths (as in Figure 51(a)). On the other hand, the maximum length of a (shared) path such that it is more cost-effective than the shortest path depends on the sharing allowed on each path.

The expressions for the maximum penalty, in terms of network delivery cost increase, from using either t_{sp} or t_{fl} for scalable delivery, for an arbitrary topology with one server and two clients are derived in appendix E. The conclusions from these results, which are key to the development of heuristic algorithms in section 7.5 are:

- the maximum penalty associated with t_{fl} is, in theory, unbounded. In practice, this bound is limited by the maximum length of a path segment in the Internet and the exact formula is provided in appendix E. The important point is that it can be very high if the load at one of the clients is significantly higher than the load at the other client. This result indicates that routing heuristics that are based only on maximizing sharing, by using the fewest number of links possible, may result in high delivery cost, compared to optimal, and thus are not reliable for routing clients in systems that use scalable protocols such as bandwidth skimming and patching.
- using t_{sp} for scalable delivery incurs the highest cost increase if the non-shared path segments in the tree are very long compared to the shortest distance between the clients. For example, in Figure 51(b), the non-shared path segments of t_{sp} have each length 10, whereas the shortest distance between the clients is 2. However, we found that, unlike t_{fl}, the maximum penalty associated with t_{sp} is bounded. In particular, an upper bound on the penalty of using t_{sp} for given server and two clients is derived as $1/(f+1) \leq 100\%$, where f is the ratio of the shortest

distance between the two clients and the sum of the lengths of the unshared segments in the tree of shortest paths. Upper bounds on the penalty for f = 1/10, 1/5, ½ and 2 are 90%, 83%, 66% and 33%, respectively. Although the bound is derived for two clients, it could be used to estimate the maximum penalty of t_{sp} with a larger number of clients by considering each pair of clients at a time.

- in spite of the bound, the maximum penalty associated with shortest path routing, in general, may be significant (more so than demonstrated in previous work). Thus, it may be worth using more sophisticated heuristics, such as the one proposed in [155]. In section 7.5, we evaluate two variants of this basic heuristic, generalized for the case of multiple replica servers, and shortest path routing, compared to the optimal solution, for a number of realistic Internet examples.

7.4.2 Insights for Replica Placement Strategies

The best previously proposed heuristic for placement of replicas in conventional CDNs is a greedy algorithm that places one server at a time trying to minimize an arbitrary cost function at each step [93]. It has also been shown that simpler rules such as placement of replicas close to the higher demand clients (*hotspot*) [115] or at the most connected nodes (*fan-out*) [85, 116] perform close to the greedy algorithm [85, 115]. This section analyzes simple topologies in search for similar kinds of insights into efficient placement strategies for scalable CDNs. In particular, we investigate whether simple rules can perform well in this scenario.

In the case of only two client nodes, placing a replica at the highest demand client is always optimal. However, for more general topologies, replica placement at the highest demand client can be significantly suboptimal. In particular, we analyzed several topologies, one of which is shown

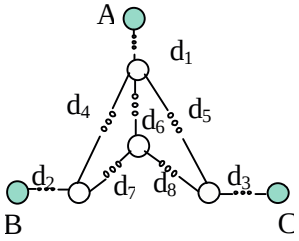


Figure 52: Canonical Topology for Placement of First Replica ($N_A \leq N_B \leq N_C$)

in Figure 52, in search for a simple rule for placement of the first replica. The topology shown in Figure 52 has three clients, *A*, *B* and *C* with demands $N_A \leq N_B \leq N_C$ and four (non-client) interior nodes. The shortest distance between each pair of nodes is shown in the figure. We found that, depending on the lengths of the path segments and on the client demands, there are scenarios where the optimal placement of the first replica is at the lowest demand client (client *A*), at the client with intermediate demand (client *B*), at the highest demand client (client *C*) or even at any of interior nodes. Analyzing these scenarios, we could only find rules for determining the optimal placement of the first replica for very specific cases. As an example, if client *C* has demand larger than the sum of the other clients' demands, i.e., $N_C > N_A + N_B$, then the first replica should be placed at client *C*. Moreover, if the clients are organized in a daisy chain, which is the case if the paths labeled with d_6, d_7 and d_8 are deleted (i.e., do not exist) from Figure 52, the optimal placement of the first replica is at the client in the center of the chain, povided that $N_C < N_A + N_B$. For the more general case, the set of rules for determining the optimal placement depends, in a complicated way, on the exact values of the demands and lengths of the path segments.

Given that we could not find a simple rule for determining the placement of the first replica that is valid for general scenarios, we use the result that the maximum cost increase, compared to optimal, of the tree of shortest paths is bounded as basis for a heuristic for first replica placement. This heuristic is described in section 7.5. In the remainder of this section, we seek insight into

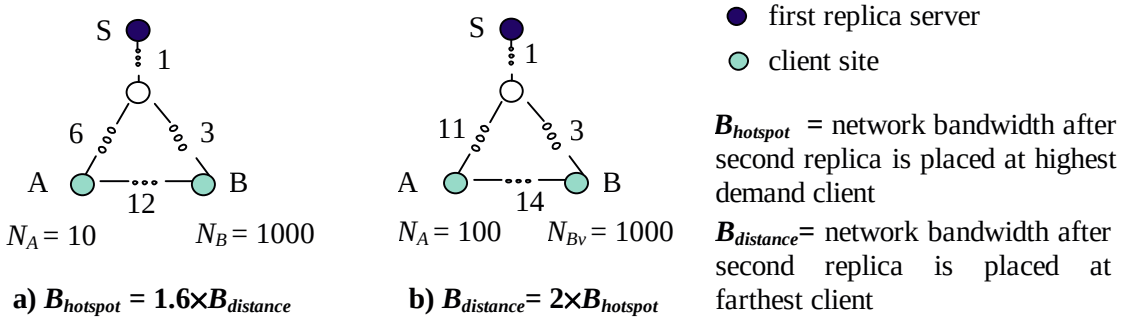


Figure 53: Comparing Algorithms for Placement of a Second Replica

efficient strategies for placement of extra replicas in a system where at least one replica has been previously placed.

Since the number of links in the delivery trees and the load on each link are the two key factors that affect the total cost of a system, we evaluate two alternative heuristics:

a) placement of replicas at the highest demand clients so as to reduce the load in the network (hotspot),

b) placement of replicas at the clients that are reached through the longest paths so as to reduce the number of links in the delivery trees (distance).

Figure 53 provides numerical examples that illustrate cases where each strategy is suboptimal. The simple topologies have two clients, *A* and *B*, with demands N_A and N_B, respectively, and a server previously placed at the top node (possibly closer to some higher demand clients, not shown in the figures). We compare the total network delivery cost after a second replica is placed in the network using one of the two strategies described above. After the second replica is placed at one of the two clients, the other client contacts the closest server for the content. Considering only the two clients shown in the figure, Figure 53(a) shows a scenario where placing the second replica at the highest demand client (i.e., client *B*) is optimal, whereas placing it at the farthest client (i.e.,

client *A*) results in a total network delivery cost increase of 60%. Figure 53(b) shows a scenario where the opposite is true and placing the replica at the highest demand client results in 100% network delivery cost increase compared to optimal[6].

These examples show that neither of two simple rules should be used for replica placement in a scalable system as the cost increase, compared to the optimal, can be very significant. In fact, it can be easily shown that the cost increase can be arbitrarily high for both strategies. For instance, one can think of a scenario where the distances from server to the two clients are approximately the same but the client that is (slightly) farther away, has a total demand much lower than the other one, in which case the distance algorithm would result in a total cost that that may be much higher than the optimal. A similar example can be constructed for the hotspot algorithm. Thus, unlike for conventional CDNs, simpler algorithms such as hotspot, fan-out or distance, are *not* good heuristics for scalable CDNs.

We point out that, intuitively, one might expect that the hotspot placement, which is based only on the client demands, does not perform as well for a scalable system as it does for a conventional system because of the different relative impact of the demands on the total delivery cost on each system. For conventional delivery, total cost increases linearly with the client demand, whereas for scalable delivery, it increases only logarithmically. Thus, the impact of placing a replica at a higher demand client in reducing the total cost is lower for scalable systems. This point is further explored in the next section.

Furthermore, these results indicate that a good heuristic for replica placement in scalable systems needs to consider both distance and demand of clients. As an example, for the simple

[6] We point out that we also experimented with placing replicas at the most connected nodes (fan-out) and found it may be significantly sub-optimal in a number of simple scenarios.

topologies in Figure 53, the optimal solution is to place the replica at the client node that has the largest product of distance to the server and the logarithm of its demand. This product corresponds to a lower bound on the bandwidth savings if the second replica is placed at that client. We will use this insight to develop a heuristic for placement that is driven by the goal of maximizing the expected bandwidth savings. This heuristic is described in section 7.5.

7.4.3 Replica Placement for Conventional and Scalable Systems

In the process of searching for a simple rule for replica placement, we derived insights into why and when the optimal placement for conventional delivery may perform so poorly if used for scalable delivery, as shown in section 7.3. In this section, we shed some light into the reasons for why this might be the case.

The key reason is the different relative impact of the demands of the clients on the cost associated with a placement for the two delivery mechanisms. For conventional delivery, the total cost associated with a placement increases linearly with both the distance from replica to clients and the demands of the clients. On the other hand, for scalable delivery, the total cost of a placement depends linearly on the number of links in the delivery tree but, for each link, only logarithmically on the demands of the clients that share that link. Thus, the effect of placing a replica close to a given client on reducing the total cost is different for conventional and scalable systems; consequently, the optimal placement might be different.

As an example, when choosing between two clients for the placement of a second replica (as in the examples in Figure 53), if the client with highest demand is the one that is farther away from all other nodes, the optimal placement is the same for both conventional and scalable delivery, i.e., at the highest demand client, to avoid routing a heavy traffic over the longer path. However, if the

lowest demand client is farther away from all other nodes, the optimal placement may be different. There is a tradeoff between placing the replica at the high demand client and routing a light traffic over a long path or placing it at the low demand client and routing a heavier traffic over a shorter path. The solution for this tradeoff might be different for conventional delivery and scalable delivery, depending on the path lengths and client demands. For instance, if conventional delivery is used, the optimal placement of the second replica is at the highest demand client in *both* topologies in Figure 53.

Note that, as illustrated in Figure 50, the reason for most of the significant differences in the optimal solutions for conventional and scalable delivery observed in our experiments lies on the solution of this tradeoff. Whereas the optimal solution for conventional delivery includes placement of replicas, mostly, at the highest demand clients (i.e., hotspot placement), the optimal solution for scalable delivery have some of the replicas placed at lower demand clients that are connected to the rest of the network through longer paths. By exploring the scenarios when the two solutions are different for a simple topology with one replica and two clients, where a second replica is placed at one of the two clients, appendix E shows that the cost increase, compared to the optimal cost, if the placement that is optimized for conventional delivery is used for scalable delivery, may be arbitrarily high.

7.5 Heuristic Near-Optimal Distribution System Design

This section proposes and evaluates a set of heuristic algorithms for computing the optimal number and placement of replica servers and routing for scalable systems. These algorithms require significantly less execution time than the optimization models and thus can be applied for designing large and heterogeneous scalable systems with near optimal delivery cost. The goal is to

produce solutions with total delivery cost very close to optimal for a broad range of network and client sites configurations.

We propose and evaluate two alternative heuristics for placing the first and subsequent replicas in a network, based on the insights drawn in section 7.4. After each replica is placed, three alternative routing heuristics are proposed to determine the near optimal delivery trees from the replicas to the client sites. The placement heuristics are greedy algorithms that places one replica at a time and keeps previous placement fixed while computing a new placement. Similarly, two routing heuristics are also greedy as the client sites are added to the delivery tree one at a time. The routing heuristics are re-executed for all client sites after each new replica is placed. As in the formal optimization models, the heuristics are derived for a single file, with a given request rate from each client site, but they could be extended to handle multiple files. The heuristics assume that the trees of shortest paths from all access points for server placement and from all client nodes to all other nodes in the network are given. Each tree can be efficiently built using Dijkstra's Algorithm, which has complexity $O(|A|+|V|\times\log(|V|))$ [19], where $|V|$ and $|A|$ are the number of nodes and arcs, respectively, in the topology.

Sections 7.5.1 and 7.5.2 describe the placement and routing heuristics, respectively, which are based on the insights drawn in the previous section, and analyze their cost complexity. The heuristics are evaluated in section 7.5.3.

7.5.1 Heuristics for Computing Replica Placement

The following observations motivate us to use the tree of shortest path as basis for determining the placement of the first replica. First, our analysis in section 7.4 of routing strategies for a simple canonical topology shows that the maximum network cost increase (compared to the optimal) in

using the tree of shortest paths for scalable delivery is bounded and, depending on the topology, may be small. Second, shortest path routing has been previously shown to perform reasonably well for routing to all client sites from one server in randomly generated networks [155]. Third, the tree of shortest paths can be built very efficiently [19]. Thus, we propose to select for placement of the first replica, the node that has the lowest total delivery cost in its tree of shortest paths to all client sites. To determine the delivery cost of a tree of shortest paths, one needs to traverse the tree from leaves to root, on a breadth first manner, visiting each node exactly once and remembering the load from each client site visited. The load and corresponding bandwidth requirements for each arc in the tree is computed as the sum of the loads from all the client sites located below that arc in the delivery tree. The cost of this algorithm is $O(|V|)$.

For placement of the i^{th} replica in the network, where $i > 1$, we propose two heuristics:

- *Trees of Shortest Paths*: select, among all unused replica access points, the one that, together with the previously placed i-1 replicas, has the lowest delivery cost in the set of trees of shortest paths when each client site receives content from the closest replica via the shortest path.
- *Maximum Savings*: select, among all unused access points that are directly connected to a node in the delivery tree(s) for the i-1 replicas, if any, the one such that the bandwidth savings from removing that node (and all its descendents) from its tree is maximized. If there are no unused access points directly connected to a node in one of the trees, use the trees of shortest paths placement algorithm.

The *trees of shortest paths* placement algorithm is a generalization of the heuristic used for first replica placement. The tree of shortest paths from each replica to all client nodes is *trimmed* by connecting each client site only to the closest replica. This algorithm iterates over the set of unused

access points for placement. At each iteration, the algorithm performs two steps. First, it determines for each client site, the closest replica among all previously placed replicas and the candidate being considered. Second, each tree of shortest paths is traversed, from the client sites (leaves) to the replica (root) on a breadth first manner, considering only the client sites that have selected that replica as the closest one. As a tree is traversed, the request rate of each visited client site is remembered, the load on each visited link is updated with the total request rate from all client sites that are below that link in the tree and the bandwidth over each link is calculated. This algorithm has order of complexity $O(|V|^2)$. At each iteration, step 1 can be executed in O(1) if the previously placed replica that is closest to each client site is known (i.e., it is computed and stored in memory at the end of each iteration); step 2 has complexity O($|V|$), as each node is visited exactly once.

Note that neither this algorithm nor the algorithm for placement of the first replica implies any specific routing algorithm. In other words, the trees of shortest paths are used only for the placement decision. Once the replica is placed, another routing heuristic may be used to build the delivery trees.

The maximum savings algorithm is based on a generalization of the simple rule derived for the simple canonical topology with two client sites in section 7.4.2: it places the next replica at the node that incurs the largest bandwidth savings if removed (together with any descendant) from its current tree. It works as follows. Let the load out of a node be the sum of the request rates of all client sites that are connected to the subtree rooted by that node. A lower bound on the bandwidth savings from removing a node from a tree is calculated by following the path from that node up to the root of the tree, subtracting from the current load on each arc visited the total load out of that

node and calculating the difference in bandwidth. This process is repeated for every unused access point for placement connected to one of the trees. Thus, it has complexity $O(|V^2|)$.

We point out that the trees of shortest path placement algorithm is potentially more efficient than the maximum savings placement because the latter gives higher priority to access points that are connected to nodes already in the tree, even when placing replicas at other access points may result in lower total cost.

More flexibility is added to the placement algorithms by allowing a newly placed replica to be moved from the root of the delivery tree to one of its internal nodes if that results in lower total cost. This is determined by reversing all links in the path from the root to that node. This extra step can be performed by traversing the delivery tree, built using one of the routing heuristics described below, from root to leaves, on a depth first manner, visiting each node exactly once. At each node i, the arc from i to one of its children in the tree is temporarily reversed. The load on the new (reversed) arc is defined as the total load out of the node i minus the load on the original arc that is currently reversed. If the bandwidth requirements for the new arc are lower than the bandwidth requirements for the original arc, the arc is permanently reversed and the traversal in the tree proceeds to visit the selected children of i. Otherwise, the traversal proceeds with the other children of i, or stops if all nodes below i have been visited. This algorithm has complexity $O(|V|)$.

7.5.2 Heuristics for Computing Delivery Trees

We propose routing heuristics for scalable streaming protocols that have required server bandwidth that is sublinear (but is not constant) with the client load, such as bandwidth skimming or patching. In particular, the algorithms developed in this section are tested in the next section for the

bandwidth skimming protocol, although we expect the results to be qualitatively the same for patching. We experiment with three heuristics:

- *Absolute MinCost*: build the delivery tree by adding one client site at a time, selecting at each step the client site that can be connected to any of the already built multicast trees with the minimum incremental bandwidth cost.
- *Ordered MinCost*: add client sites in order of decreasing load (i.e., logarithm of the request rate N_i) and, in case of tie, increasing distance to the closest server. Each client site is connected to one of the trees by the path with minimal incremental bandwidth cost (as in the *absolute mincost* algorithm).
- *Shortest path:* we also consider shortest path routing as it was found to perform reasonably well for one server [155].

The *absolute mincost* heuristic was proposed in [155] for routing from one server to every client site and is here generalized for multiple servers. It works as follows. At each iteration, it considers all client sites that have not been connected to the tree yet. For each one of those client sites, it determines the path that connects it to any of the trees with minimum incremental bandwidth cost. This path is determined by traversing the trees from root to leaves and at each visited node, calculating how much bandwidth cost would be added to the current cost associated with the tree if the client site is connected to that node through the shortest path. In visiting each node *i* of the tree, the incremental bandwidth cost over the path connecting node *i* to the root is part of the incremental bandwidth cost associated with connecting the new client site to each descendant of node *i* in the tree. Thus, each node of the tree can be visited exactly once to determine the minimum incremental bandwidth path for a client site. This has complexity O(|*V*|). The procedure is repeated, at each iteration, for all client sites that have not been connected yet, and

one client site is added to a tree at the end of each iteration. Thus, the complexity of the algorithm is $O(|V|^3)$.

The *ordered mincost* algorithm works similarly to the *absolute mincost* algorithm, except that the order on which the client sites are added to the trees is predefined. Client sites with higher request rates are added first in an attempt to reduce the delivery cost by serving the client sites that impose higher load on the system with the "best" paths. Because the order of addition of client sites is independent of the routing decision, the complexity of the *ordered mincost* algorithm is $O(|V|^2)$.

The routing heuristics described above were developed for protocols, such as bandwidth skimming and patching, that have per-hop network bandwidth that depends on the number of clients that are served over each hop. In [67], the authors propose heuristic algorithms for constructing near optimal delivery trees for a single live multicast. Their heuristics rebuild the delivery tree every time a client joins or leaves the multicast group. Like for live streaming, the per-hop network bandwidth for periodic broadcast is also independent of the number of clients that are served over that hop. In that case, unlike for bandwidth skimming and patching, the optimal routing is known a priori, it is the tree of fewest links. Although the optimal tree of fewest links cannot be optimally computed efficiently, a heuristic is proposed in [155]. However, we point out that the optimization model for periodic broadcast is much more efficient than the model for bandwidth skimming, and thus is expected to be practical for larger systems. Experimenting with the optimization model and heuristics for periodic broadcast is left for future work.

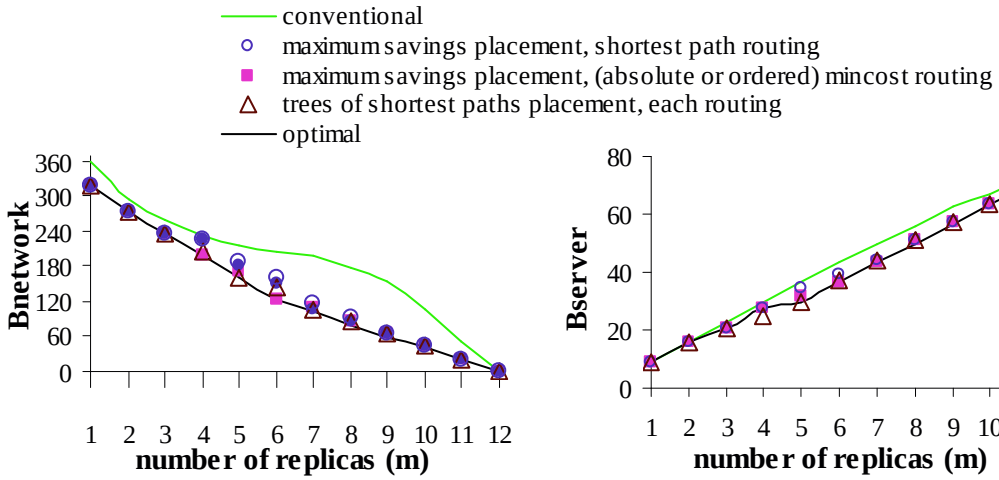
7.5.3 Performance of the Heuristics

This section evaluates the performance of the heuristic algorithms using the set of AS and router level topologies defined in Table 21 (section 7.3). The total delivery cost of the systems obtained

with each algorithm is compared against the optimal cost, obtained with the optimization model. We consider six different heuristic algorithms, created by combining one of the two heuristics for placement of an extra replica with one of the three routing heuristics. In the figures below, we will refer to each case by stating the heuristic used for placement and the heuristic used for routing.

After analyzing all the AS and router level topologies with different client load heterogeneity and dispersion considered in section 7.3, we found that all six heuristics result in solutions with approximately the same delivery cost for the topologies with lower degree of client dispersion. However, for router level topologies with higher degree of client dispersion (client sets covering 2-4 continents), the differences among the costs of the heuristic solutions are somewhat more unequal (but are still within 30% of each other). Furthermore, we found that *trees of shortest paths* placement combined with *ordered mincost* routing results in better solutions, which are within 16% of optimality for all systems considered.

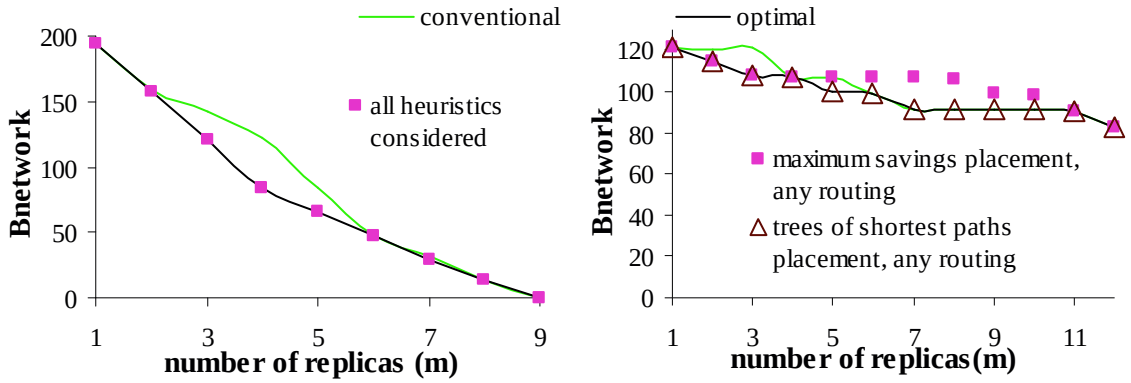
Figure 54 shows the results for all six heuristics for three example topologies. For comparison purposes, we also show the curves of the optimal cost, as well as the cost if the optimal solution for conventional delivery is used for scalable delivery. Figures 54(a)-(b) show the near optimal network and server bandwidth for a topology. Note that the solutions obtained with the *trees of shortest paths* placement heuristic are the same regardless of which routing algorithm is used. Furthermore, if the *maximum savings* placement heuristic is used, the solutions are the same for *absolute* and *ordered mincost* routing. Figures 54(c)-(d) shows the network bandwidth for two other systems. The curves for server bandwidth are omitted because they are the same for all solutions. In Figure 54(c), all six heuristics produce systems with the optimal cost and in Figure 54(d), the cost of the solution depends only on the placement algorithm. Figure 55 shows the (very similar) systems produced by the heuristics as well as the optimal system for conventional



a) Network Bandwidth ($B_{network}$)

b) Server Bandwidth (B_{server})

(8 client sites in US and 1 in Canada : N_i = 1000; 3 client sites in Europe: N_i = 100)
(CMU, SDSC, SU, UCB, UW, TX, UO, CU ; GU-UK , SE-FR, UC-GH)
(router level topology)

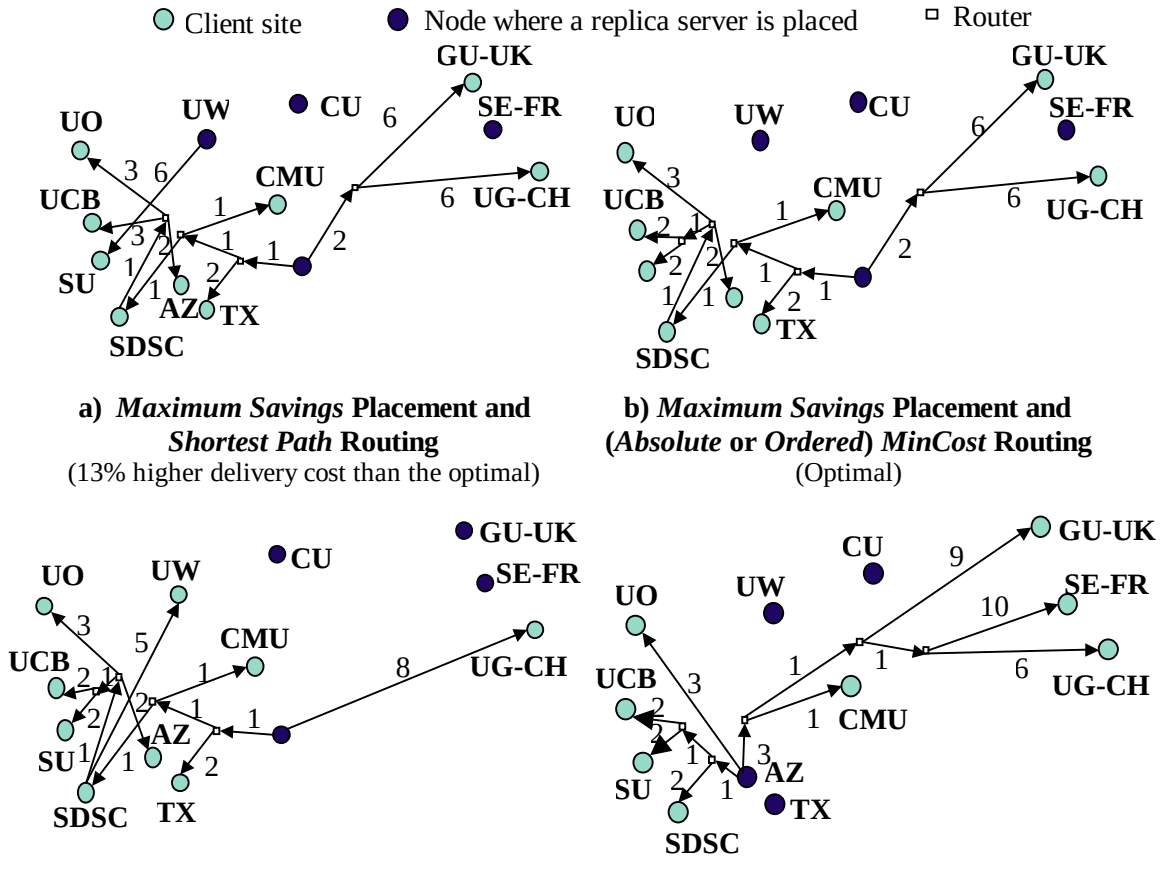


c) 9 heterogeneous client sites in US
(4 clients: N_i=1000; 5 clients : N_i=100)
(UW, UCB, SDSC, SU; UO,TX,CMU, AZ,VY)
(router level topology)

d) 12 homogeneous client sites in US, Canada and Europe
(N_i=1000: AZ,CMU,SDSC,SU,UCB, UW, TX, UO, CU,GU-UK, SE-FR, UC-GH)
(AS level topology)

Figure 54: Example Cost Comparison of Alternative Heuristic Algorithms for Replica Placement and Routing

unicast delivery for the topology in Figure 54 and 4 replicas. The main conclusions from these results are:



a) *Maximum Savings* Placement and *Shortest Path* Routing
(13% higher delivery cost than the optimal)

b) *Maximum Savings* Placement and (*Absolute* or *Ordered*) *MinCost* Routing
(Optimal)

c) *Trees of Shortest Paths* Placement and *Shortest Path* Routing or (*Absolute* or *Ordered*) *MinCost* Routing
(3% higher delivery cost than the optimal)

d) Conventional
(16% higher delivery cost than the optimal)

Figure 55: Alternative Scalable Distribution Systems
(router level topology, m = 4 replicas, γ= 0, $N_{\text{GU-UK}} = N_{\text{SE-FR}} = N_{\text{UC-GH}} = 100$, $N_{\text{others}} = 1000$)

- The overall performance of all heuristics is very good for homogeneous, heterogeneous, router and AS topologies. For all the validation experiments, the best heuristic results in solutions within 16% of optimal for both total server and network delivery cost. However, in many scenarios, they result in systems with optimal cost, as in Figure 54(c).

- The *ordered mincost* routing heuristic performs just as well as the more complex *absolute mincost* for all systems analyzed, with the advantage of being simpler. Furthermore, we find

that *shortest path* routing may incur higher cost increase than previously observed in [155], depending on the placement algorithm used. In our experiments, we observed a cost increase of up to 28% if shortest path routing is used together with *maximum savings* placement (Figure 54(a), 6 replicas).

- The cost increase due to *shortest path* routing observed in our results is still somewhat more modest than one might expect. In order to understand why this is the case, we inspected several topologies and solutions and found that most of the non-shared segments of the shortest paths to the client sites are not much longer than the shortest distance between the client sites. As an example, in Figure 55(a), the shortest distance between client sites GU-UK and UG-CH is 11 and the non-shared segments of the shortest paths have, each, length 6. Based on the upper bound derived in appendix E, we conjecture that this is the reason for the relatively low cost increase associated with *shortest path* routing.
- The *maximum savings* placement algorithm is not always efficient (as shown in Figures 54(a), (b), (d)). On the other hand, as expected, the *trees of shortest paths* placement heuristic performs better across all topologies, especially for AS topologies (Figure 54(d)), with a maximum cost increase observed of only 16%, compared to the optimal, for the systems considered. Furthermore, for all topologies considered, the performance of this heuristic is usually approximately the same regardless of the routing heuristic used. In particular, its performance is much less sensitive to the use of *shortest path* routing than the *maximum savings* placement algorithm. This is particularly interesting in case our algorithms are implemented on top of IP multicast.

Based on these results, the algorithm that uses the *trees of shortest paths* placement combined with the *ordered mincost* routing delivers the best performance, resulting in systems with total

network delivery cost within 16% of optimality for all configurations tested. Similar accuracy was also observed for the total server delivery cost. This approximate algorithm is a good compromise between accuracy and efficiency and we expect it to be a valuable tool for designing larger and more complex scalable systems with close to optimal delivery cost. Experimenting with larger topologies is subject for future work.

We point out that solving the optimization models require significant more execution time not only to find the optimal solution but also to reduce the searching space to solutions that were within the same accuracy (i.e., 16%) of the solutions obtained with the best heuristic. The optimal solution time may be significantly reduced by using the heuristic solution as starting point for the searching procedure. Experimenting with this hybrid strategy is a direction for future work. Furthermore, we also point out that the algorithms used for solving the optimization models provide, for any intermediate solution found in the searching procedure, an upper-bound on the gap between the cost of that solution and the optimal cost. Thus, one interesting approach to determine the accuracy of a heuristic solution for larger sytems, for which the truly optimal solution can not be obtained, is to use the heuristic solution as starting point for the searching procedure in the model and use the upper-bound on the gap, provided by the solver, as an estimate of the accuracy of that solution.

As a final note, we also experimented with fan-out placement (i.e., placement at the most connected nodes) both for the first replica and for an extra replica. Fan-out placement is attractive because it is simple, does not rely on knowledge about the client sites and it was shown to perform well for conventional systems [85, 116]. However, we found that placing replicas based only on the connectivity of the nodes can be significantly more sub-optimal for scalable delivery (cost increase of up to 200% compared to optimal for the topologies given in Table 21). Thus, this section showed only the results for the more competitive heuristics.

7.6 Summary of the Chapter

This chapter developed exact and approximate methods for finding the optimal number and placement of replicas of one object and the optimal routing for scalable delivery, which can be applied to a number of different scalable multicast streaming protocols. We developed a CDN design model that includes more precise estimates of network bandwidth and applied the model to show that, for a number of realistic Internet topologies, using placement and routing strategies that are optimized for conventional unicast delivery to scalable CDNs can result in significant cost increase (over 100% increase in network bandwidth in some cases), compared to optimal, especially if client sites have heterogeneous loads and bandwidth is charged on a per router basis.

We also analyzed simple topologies to draw insights into efficient approximate algorithms for placement and routing for scalable CDNs. In particular, we showed that, unlike for conventional CDNs, simple rules of thumb such as placement of servers near the client sites that have highest demand [115] or at the most connected nodes [85], do not perform well for scalable delivery. We also derived an upper bound for the cost increase associated with the use of shortest path routing, the basis for the routing algorithms in today's Internet, for scalable delivery for a simple canonical topology with two client sites.

These insights led to the development of six heuristic algorithms, which require significantly less execution time and are thus applicable to large and heterogeneous systems. We found that the best heuristic, which uses the *trees of shortest paths* algorithm for placement and *ordered mincost* for routing, results in systems with total delivery cost within 16% of optimality for all configurations analyzed.

Chapter 8

Conclusions and Future Work

This thesis has explored the design of streaming media content distribution networks with (close to) minimum delivery cost, where the delivery cost includes server and network costs. We developed caching algorithms for conventional CDNs, design models to determine the optimal server content, server placement and routing for scalable CDNs and approximate algorithms for server placement and routing in scalable CDNs. Our methods take into consideration the special characteristics of streaming media, in particular, the high bandwidth requirements, and address the new and complex tradeoffs introduced by sharing of multicast streams among multiple clients. In order to evaluate our caching algorithms and CDN design models, we identified a reasonably small set of derived system and workload parameters that have primary impact on CDN delivery cost. In our experiments, these parameters are varied over a wide range of values in order to evaluate the solutions over as much of the system design space as it is practically feasible. We have also characterized two streaming media workloads more completely than previous workloads have been characterized with the goals of gaining insight into efficient caching strategies, understanding how client request rate and media file access frequency vary with time, and providing data for generating synthetic workloads.

Sections 8.1 – 8.3 summarize the main contributions and results regarding the analysis of the two media server workloads, the storage strategies for conventional and scalable CDNs and the server placement and routing strategies developed for scalable CDNs. Section 8.4 provides possible directions for future work.

8.1 Workload Analysis

We have provided an extensive analysis of the workloads of two educational streaming media servers in operation in two major universities in the United States: the eTeach server, at the University of Wisconsin-Madison and the BIBS server, at the University of California-Berkeley. Our analysis provided a more complete coverage of the workload characteristics than previous studies. New aspects of the workload characterization included: (a) distribution of overall and per-file request inter-arrival times during periods of approximately stationary request rate, (b) distribution of media delivered per session, during periods of approximately stationary average media delivered per session, (c) changes in relative file access frequencies from one day to another and from one hour to the next, (d) distribution of file access frequency per day and during periods of approximately stationary file access frequency, (e) distribution of access frequency to each ten-second segment of a file, for files with different (overall) access frequency rank, (f) fraction of the new content (file and file segments) requested during each hour that is requested only once during the hour, where a new content is an object that was not requested in the previous two, four or eight hours, and (g) distribution of the time until the next access to new content. Some of the characteristics (e.g., media per session, file access frequency) were analyzed separately for different file size ranges, showing that accurate distributions depend on the file size. The collection of distributions provides useful data for generating realistic synthetic workloads.

We found a high degree of similarity in the measures that could be analyzed for both servers. That is, a large fraction of the eTeach requests and the BIBS sessions have short duration, file access frequency on each server can be modeled with the concatenation of two Zipf-like distributions, and the distribution of media delivered per session (or per request) depends on the file size and is heavier tailed for longer files. The only characteristic measured in both servers for which we found a difference was the distribution of session inter-arrival times during periods of approximately stationary session arrival rate. On BIBS, inter-session times are better modeled with an exponential distribution, whereas in eTeach, they appear to be better modeled with a heavier tailed distribution. We note, however, that session arrivals are not identified in the eTeach log. Thus, new session requests in eTeach were identified heuristically.

The analysis of the media server workloads led to several important insights for designing efficient streaming CDNs. First, we found that a significant fraction (60%-85%) of the new files (or new file segments) accessed during an hour are accessed only once in that hour. Furthermore, 50-80% of them are not accessed again in the next eight or even sixteen hours in the future. This implies that caching new content every time a request misses in the cache, without knowledge of its access frequency, may result in significant wasted disk write bandwidth and, consequently, in poor cache performance.

Second, we found that, in eTeach, the distribution of accesses to each ten-second segment is approximately uniform for the most frequently accessed files in each of ten different days we examined. This implies that, in spite of the large number of requests to small portions of the files, the access frequency for each of the most frequently accessed files, measured, for instance, for the requests that start at the beginning of the files, might be used to estimate segment access frequency for those files. It also implies that, for the most popular files, full file caching might be the best

strategy in a CDN that uses unicast delivery. Furthermore, for each of the less frequently accessed files we examined, we found that the distribution of accesses to each ten-second segment is skewed, with more frequent requests for earlier segments. In practice, one could keep access counts for two or three positions of the file (e.g., 0, 10%, 50% of the file) to determine whether the distribution of access frequency is uniform or skewed towards earlier segments. Further work is needed to examine more files, to determine whether these observations hold more generally.

Third, we found that relative file access rates change significantly from day to day and even from one hour to the next (especially on BIBS). For instance, we observed variations in a file access frequency rank from 1st to 10th from one hour to the next. Wider variations were also observed. A CDN configuration (e.g., server content or routing) may need to be updated in response to those changes.

Our analysis also provided example profiles of client interactive behavior, including the start position and end position of client requests for files with different access frequencies. We collected profiles for the three most frequently accessed files in ten different days in eTeach. The information available in these profiles, i.e., the range of media file data retrieved in each request, can be used, as starting point, for developing client interactive access models and the formulas for server bandwidth requirements for these access models.

Finally, despite the large number of interactive requests to very short segments of the files, we found that the bandwidth savings if multicast streaming were used to deliver the three most frequently accessed files in eTeach would be up to 30%. Further bandwidth savings may be achieved if the server does not interrupt delivery every time a client pauses.

8.2 Server Content

For conventional unicast CDNs, using simulation with synthetic workloads, we found that, in practical terms, the simple CMP caching policy, which stores the files with the highest measured access frequencies, outperforms the best previously known caching algorithm, RBC, which stores a mixture of full files and intervals and is much more complex than CMP. In order to further analyze interval caching, we proposed and evaluated two other new policies, called Pooled RBC and CMP/IC. Pooled RBC improves on the original RBC by sharing the allocation of proxy disk bandwidth among multiple files. The simpler CMP/IC is a hybrid policy that uses the CMP policy plus interval caching performed in a separate cache, either in main memory or on disk. Comparing Pooled RBC, CMP/IC and CMP, we found that CMP/IC has performance at least as good as Pooled RBC, with the advantage of being much simpler to implement. However, the even simpler CMP, which does not include interval caching, delivers competitive performance compared with the other two policies. These results led to the conclusion that the benefit of interval caching for unicast streaming is, at best, modest. Furthermore, since interval caching can be expected to have even more limited performance benefit for scalable streaming, it does not need to be considered for designing scalable CDNs.

Determining the server content that minimizes the total delivery cost for scalable CDNs is considerably more complex due to the new tradeoffs introduced by cost sharing of multicast streams among multiple clients. We have proposed three new modifications to the scalable multicast bandwidth skimming protocol that enable prefix caching at proxy servers. For each modification, we developed simple optimization models with approximate network bandwidth costs for determining the minimum cost CDN design. We have applied the optimization models to gain insight into the optimal proxy storage strategy over a large region of the CDN design space.

We used these models to evaluate the impact of different protocol optimizations, including batching versus non-batching of client requests for suffix streams from the origin, unicast versus multicast origin streaming and different client receive bandwidth, on the optimal CDN delivery cost. This evaluation covered a much wider and more realistic CDN configuration space and used a more efficient protocol (i.e., bandwidth skimming) than previous work, leading to new results for designing scalable CDNs.

One key conclusion is that storing content at multicast enabled proxy servers is only cost effective if (a) the origin server uses unicast or (b) the client request rate is very low, in which case multicast is not very effective, or (c) the average cost of a proxy stream is a small fraction of the average cost of an origin stream (approximately $1/P$ or less, where P is the number of proxy servers).

Furthermore, we found that, unlike previous results would suggest, prefix caching and batching is cost-effective under realistic proxy bandwidth assumptions and for the bandwidth skimming protocol only in the small region of the design space defined by (a) multicast origin streaming, (b) large number of proxy servers and (c) small per-file request rate at each proxy ($N/P \approx 1$). In all other regions of the design space, the simpler bandwidth skimming protocol without batching and an all-or-nothing storage strategy (nearly) minimize total delivery cost. Moreover, we found that, unlike conventional CDNs, where the optimal proxy content consists of the most frequently requested files, the optimal proxy content for scalable CDNs include fewer and less popular files as the total request rate increases and/or as the ratio of the average cost of a proxy stream to the average cost of an origin stream increases.

We also found that, if clients have limited receive bandwidth (e.g., if they can listen to only 1.2 simultaneous streams), which might be the case if they request high quality videos over a non-

broadcast network, the total delivery cost, compared to the optimal cost if they can receive at least two simultaneous streams, can be significantly higher. For example, for CDNs with P=10 proxy servers, request arrival rate at each proxy N/P = 10000 and average cost of a proxy stream that is 10% of the average cost of an origin stream (i.e., β = 0.1), the increase in delivery cost is up to 200% (for large cache sizes).

Finally, if the origin uses multicast, instead of unicast delivery, the total CDN delivery cost is greatly reduced. For instance, for a CDN with ten proxy servers, overall request rate at each proxy M/P = 1000 and $\beta \leq 0.3$, the bandwidth savings if the origin uses multicast delivery is around 85%. This observation should encourage a wider deployment of IP or application level multicast protocols in the Internet.

8.3 Server Placement and Routing Strategies

We have also developed more detailed CDN design optimization models, which include more precise network bandwidth costs than in previous models, and approximate algorithms to determine the (near) optimal number and placement of replicas of an object and the delivery tree topologies from those servers to a given set of client sites, with corresponding request rates for the object. Our CDN design models include the network bandwidth required for each hop in the delivery trees and can be applied to a number of different scalable streaming protocols, including bandwidth skimming, patching and periodic broadcast, as well as to conventional unicast streaming, by using the appropriate (server and network) bandwidth formulas for each protocol. We point out that, the solution of the model that uses the bandwidth formulas for periodic broadcast consists of, in addition to the optimal server placement, the set of delivery trees with fewest links, which is the optimal routing strategy for that protocol.

The models were applied to show that, for a number of practical and realistic Internet topologies, if placement and routing strategies that are optimized for conventional delivery are used for a CDN that uses a scalable delivery protocol (in particular, the bandwidth skimming protocol), the total delivery cost can be significantly higher than if the optimal placement and routing for the scalable CDN are used. In particular, the network cost increase is over 100% in some cases when client sites have heterogeneous request rates and bandwidth is charged on a per router basis.

We analyzed simple canonical topologies to draw insights into efficient approximate algorithms for placement and routing for scalable CDNs. We derived an upper bound on the cost increase associated with the use of shortest path routing, the basis for the routing algorithm in today's Internet, for scalable delivery. This upper bound, derived for two client sites, is $1/f + 1$, where f is the ratio of the shortest distance between the two clients and the sum of the lengths of the unshared segments in the tree of shortest paths from server to clients. We also showed that, unlike for conventional CDNs, simple rules of thumb, such as placement of servers close to the client sites that have highest demand [115] or at the most connected nodes [85], do not perform well for scalable delivery.

Based on these insights, we proposed and evaluated six efficient heuristics, which require significantly less execution time than solving the optimization model and are thus applicable to large and heterogeneous systems. We found that, for a number of Internet examples, the best heuristic consists of (1) placing servers on the nodes that have the lowest total cost for their tree(s) of shortest paths to the clients, assuming clients send requests to the closest server, and (2) building the actual delivery tree(s) by adding clients in decreasing order of site load and connecting each client site to a server using the path that has minimum incremental bandwidth cost. This heuristic

algorithm results in systems with total delivery cost within 16% of optimality for a variety of Internet examples examined.

8.4 Future Work

There are many possible directions for future research suggested by the results in this thesis, including:

1. Analysis of other recent educational media workloads and other types of workloads, such as entertainment server workloads. It would be interesting to determine whether the observations for the eTeach and BIBS workloads in this thesis hold for other workloads. A more precise characterization of the time varying aspect of client request rates and file access frequencies for these workloads would also be worthwhile. It would also be worthwhile to develop a more complete characterization of client interactive requests, including the range of data (i.e., start and end positions in the file) retrieved by each request. In particular, one could determine the distribution of start positions for interactive requests and the distribution of end positions, conditioned to the start position, for a large variety of files, with different sizes and relative access frequencies. These distributions can be used to develop client interactive access models and formulas for the server bandwidth requirements for these models. Some initial models and corresponding bandwidth calculations were developed in [88, 135]. The most general previously proposed access model assumes arbitrary start and end positions for independent interactive requests [135]. The authors found that, under these assumptions, the server bandwidth requirements increase with the square root of the client request rate [135]. New models and bandwidth formulas derived from the actual distributions of start and end positions observed for different files should be compared against these previous formulas to determine whether more sophisticated models are necessary. In particular, one might consider exploring

whether there is significant temporal locality among the interactive requests and how it affects the server bandwidth calculations.

2. Evaluation of alternative measures of current client request rates. There are several proposals in the literature, including (a) the inverse of a weighted average of the most recent inter-request time and the mean inter-request time computed when the previous request arrived [138], and (b) for any integer k, the ratio of k to the time since the k^{th} most recent client request (used as the measure in the *LRU-k* policy [106]). These and other estimates should be compared in terms of their accuracy during periods of approximately stationary behavior, how fast they identify changes in the request rate, and their time and memory complexity. In addition to the previously proposed measures, this comparison could include methods for change point detection [40], which are used in economics, finance and signal processing to detect changes in the parameters of the distribution of a sequence of observations.

3. Extensions to the CDN design models to include other types of heterogeneity including heterogeneous object access frequencies per proxy and to include bandwidth formulas that account for client interactive requests (as discussed in 2). Applying the models to compare the insights obtained against the results shown in this thesis is also a direction to pursue.

4. Development of simple rules of thumb for determining the (near) optimal proxy content for scalable CDNs. For example, such rules of thumb could specify a range of access frequencies for the files that should be stored at each proxy server. Although the design models for determining the optimal proxy content for homogeneous workloads can be solved relatively quickly, it might be possible to develop simple and more efficient rules of thumb that can be applied as client requests arrive and that result in near optimal systems. Simple rules of thumb

would be especially useful for heterogeneous workloads and proxy capacities, in which case we expect the solution time of design models to be longer due to the higher complexity.

5. Implementation of methods (i.e., design models and/or new rules of thumb) for determining the (near) optimal proxy content in the SWORD prototype [101]. The SWORD prototype, being developed at the University of Wisconsin-Madison, consists of two modules, a server module and a client module, and implements the bandwidth skimming protocol for delivering content from the server module to a number of client modules. It is currently installed in the eTeach system and is being tested under a live workload consisting of students retrieving the eTeach lectures. Experimenting with proxy caching in the prototype would provide a valuable opportunity to test the efficiency and performance of the methods, especially if it is used to serve a more widely distributed live workload. One key issue that must be addressed to perform these experiments is the design of an efficient online implementation of our methods (including a measure of client request rates, as discussed in 3), with minimum overhead added to the prototype.

6. Extensions to the placement and routing approximate algorithms to solve for multiple objects concurrently, allowing varying constraints on sharing of placement and routing among the objects. The design models and heuristics in this thesis can be applied to multiple objects by solving them separately for each object or each category of object, where a category is defined by “similar” client request rates, and then defining a method for merging the solutions in order to obtain a global solution with (near) optimal delivery cost. However, an algorithm that determines the placement and routing for multiple objects concurrently may result in CDNs with total delivery cost closer to optimal.

7. Evaluation of the CDN design models for periodic broadcast streaming as well as development and evaluation of approximate algorithms for periodic broadcast streaming. Heuristics for routing for periodic broadcast were developed in [67]. Furthermore, the heuristic to build the tree of fewest links proposed in [155] seems appropriate for periodic broadcast since the tree of fewest links is the optimal delivery tree for this protocol. However, efficient heuristics for server placement for periodic broadcast are unknown. One possible heuristic to consider is, for instance, to place servers at the nodes that have the lower total delivery cost in the trees built using the heuristic proposed in [155]. This heuristic as well as any other new algorithm should be compared against the simpler trees of shortest paths placement algorithm, developed, in this thesis, for scalable protocols that have bandwidth requirements sublinear (but not constant) with the client request rates (e.g., bandwidth skimming, patching).

Bibliography

[1] M. Abrams, C. R. Standbridge, G. Abdulla, S. Williams and E. A. Fox, "Caching Proxies: Limitations and Potentials", *Proceedings 4th International WWW Conference*, pp. 119-133, Boston, Massachusetts, December 1995.

[2] S. Acharya and B. Smith, "An Experiment to Characterize Videos on the Web", *Proceedings Multimedia Computing and Networking (MMCN)*, San Jose, CA, January 1998.

[3] S. Acharya and B. Smith, "MiddleMan: A Video Caching Proxy Server", *Proceedings 10th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, Chapel Hill, NC, June 2000.

[4] S. Acharya, B. Smith and P. Parnes, "Characterizing User Access to Videos on the World Wide Web", *Proceedings SPIE/ACM Conference on Multimedia Computing and Networking (MMCN)*, San Jose, CA, January 2000.

[5] C. C Aggarwal, J. L. Wolf and P. S. Yu, "A Permutation Based Pyramid Broadcasting Scheme for Video On-Demand Systems", *Proceedings IEEE International Conference on Multimedia Computing and Systems (ICMCS)*, pp. 118-rovi, Hiroshima, Japan, June 1996.

[6] W. Aiello, F. Chung and L. Lu, "A Random Graph Model for Massive Graphs", *Proceedings 32nd Annual ACM Symposium on Theory of Computing*, Portland, OR, May 2000.

[7] "Akamai Website", http://www.akamai.com/index_flash.html .

[8] R. Albert and A. Barabasi, "Topology of Evolving Network: Local Events and Universality", *Physical Review Letters*, vol. 85, no. 24, pp. 5234-5237, December 2000.

[9] J. M. Almeida, D. L. Eager, and M. K. Vernon, "A Hybrid Caching Strategy for Streaming Media Files", *Proceedings Multimedia Computing and Networking* (*MMCN*), San Jose, CA, January 2001.

[10] J. M. Almeida, J. Krueger, D. L. Eager, and M. K. Vernon, "Analysis of Educational Media Server Workloads", *Proceedings 11th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, Port Jefferson, NY, June 2001.

[11] J. M. Almeida, D. L. Eager, M. C. Ferris, and M. K. Vernon, "Provisioning Content Distribution Networks for Streaming Media", *Proceedings IEEE INFOCOM*, pp. 1746-1755, New York, NY, June 2002.

[12] V. Almeida, A. Bestavros, M. Crovella and A. de Oliveira, "Characterizing Reference Locality in the WWW", *Proceedings IEEE International Conference in Parallel and Distributed Information Systems*, Miami Beach, FL, December 1996.

[13] E. J. Anderson, T. E. Anderson, S. D. Gribble, A. R. Karlin and S. Savage, "A Quantitative Evaluation of Traffic-Aware Routing Strategies", *Proceedings ACM SIGCOMM*, San Diego, CA, August 2001.

[14] M. Andrews, B. Shepherd, A. Srinivasan, P. Winkler, F. Zane, "Clustering and Server Selection using Passive Monitoring", *Proceedings IEEE INFOCOM*, New York, NY, June 2002.

[15] J. Apostolopoulos, T. Wong, W-T. Tan, S. Wee, "On Multiple Description Streaming with Content Delivery Networks", *Proceedings IEEE INFOCOM*, pp. 1736-1745, New York, NY, June 2002.

[16] M. Arlitt, L. Cherkasova, J. Dilley, R. Friedrich and T. Jin, "Evaluating Content Management Techniques for Web Proxy Caches", *Proceedings 2nd Workshop on Internet Server Performance,* Atlanta, GA, May 1999.

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

[18] M. Arlitt and C. Willliamson, "Web Server Workload Characterization: The Search for Invariants", *Proceedings. ACM Sigmetrics,* Philadelphia, PA, May 1996.

[19] M. Barbehenn, "A Note on the Complexity of Dijkstra's Algorithm for Graphs with Weighted Vertices", *Transactions on Computers*, vol. 47, no. 2, pp. 263, February 1998.

[20] P. Barford and M. Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation," *Proceedings Performance/ACM Sigmetrics*, Madison WI, June 1998.

[21] A. Bar-Noy and R. E. Ladner, "Competitive On-line Stream Merging Algorithms for Media-on-Demand", *Proceedings 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp.364-373, Washington, DC, January 2001.

[22] P. Barford, A. Bestavros, A. Bradley and M. Crovella, "Changes in Web Client Access Patterns: Characteristics and Caching Implications", in *World Wide Web Journal, special issue on Characterization and Performance Evaluation,* December 1998.

[23] "Berkeley Internet Broadcasting System", http://bmrc.berkeley.edu/bibs .

[24] E. Bommaiah, K. Guo, M. Hofmann, S. Paul, “Design and Implementation of a Caching System for Streaming Media over the Internet”, *Proceedings 6th IEEE Real Time Technology and Applications Symposium (RTAS)*, pp. 111-121, Washington, DC, May/June 2000.

[25] L. Breslau, P. Cao, L. Fan, G. Phillips, S. Shenker, “Web Caching and Zipf-like Distributions: Evidence and Implications”, *Proceedings IEEE INFOCOM*, New York City, NY, March 1999.

[26] A. Broido, Kc Claffy, "Internet Topology: Connectivity of IP Graphs", *Proceedings SPIE International Symposium on Convergence of Information Technologies and Communication (ITCom)*, Denver, CO. August 2001.

[27] A. Brooke, D. Kendrick and A. Meeraus, *GAMS: A User’s Guide*, The Scientific Press, South San Francisco, CA, 1988.

[28] D. W. Brubeck and L. A. Rowe, “Hierarchical Storage Management in a Distributed VOD System”, *IEEE Multimedia*, vol. 3, no. 3, pp. 37-47, Fall 1996.

[29] T. Bu and D. Towsley, “On Distinguishing between Internet Power Law Topology Generators”, *Proceedings IEEE INFOCOM*, New York, NY, June 2002.

[30] J. W. Byers, J. Considine, M. Mitzenmacher and S. Rost, “Informed Content Delivery across Adaptive Overlay Networks”, *Proceedings ACM SIGCOMM*, pp. 47-60, Pittsburgh, PA, August 2002.

[31] J. Byers, M. Luby, M. Mitzenmacher and A. Rege, “A Digital Fountain Approach to Reliable Distribution of Bulk Data”, *Proceedings ACM SIGCOMM*, Vancouver, Canada, September 1998.

[32] “CAIDA Website”, http://www.caida.org/ .

[33] Y. Cai, K. A. Hua and K. Vu, "Optimizing Patching Performance", *Proceedings IS&T/SPIE Conference on Multimedia Computing and Networking (MMCN)*, pp. 204-215, San Jose, CA, January 1999.

[34] K. Calvert, M. Doar and E. W. Zegura, "Modeling Internet Topology", *IEEE Communications Magazine*, vol. 35, no. 6, pp. 160-163, June 1997.

[35] P. Cao and S. Irani, "Cost-aware WWW Proxy Caching Algorithms", *Proceedings USENIX Symposium on Internet Technology and Systems*, pp. 193-206, Monterey, California, December 1997.

[36] R. L. Carter and M. E. Crovella, "Dynamic Server Selection Using Bandwidth Probing in Wide-Area Networks", *Proceedings IEEE INFOCOM*, Kobe, Japan, April 1997.

[37] Y. Chae, K. Guo, M. M. Buddhikot, S. Suri and E. W. Zegura, "Silo, Rainbow, and Caching Token: Schemes for Scalable Fault Tolerant Stream Caching", *IEEE Journal on Selected Areas in Communications on Internet Proxy Services*, vol. 20, no. 7, September 2002.

[38] S.-H. G. Chan and F. A. Tobagi, "Caching Schemes for Distributed Video Services", *Proceedings IEEE International Conference Communications (ICC)*, pp. 994-1000, Vancouver, Canada, June 1999.

[39] S.-H. G. Chan and F. A. Tobagi, "Designing Hierarchical Storage Systems for Interactive On-Demand Video Services", *Proceedings IEEE Multimedia Applications, Services and Technologies*, Vancouver, Canada, June 1999.

[40] J. Chen and A. K. Gupta, Parametric Statistical Change Point Analysis, Birkhäuser (Architectural), May 2000.

[41] Q. Chen, H. Chang, R. Govindan, S. Jamin, S. Shenker, W. Willinger, "The Origin of Power Laws in Internet Topologies Revisited", *Proceedings IEEE INFOCOM,* New York, NY, June 2002.

[42] L. Cherkasova and M. Gupta, "Characterizing Locality, Evolution and Life Span of Accesses in Enterprise Media Server Workloads", *Proceeding 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV),* pp. 33-42, Miami, FL, May 2002.

[43] L. Cherkasova and M. Karlsson, "Dynamics and Evolution of Web Sites: Analysis, Metrics and Design Issues", *Proceedings 6th International Symposium on Computers and Communications (ISCC'01),* Hammamet, Tunisia, July 2001.

[44] M. Chesire, A. Wolman, G. Voelker and H. Levy, "Measurement and Analysis of a Streaming Media Workload", *Proceedings 3rd USENIX Symposium on Internet Technologies and Systems*, San Francisco, CA, March 2001.

[45] Y.-H. Chu, S. G. Rao nd H. Zhang, "A Case for End System Multicast", *Proceedings ACM Sigmetrics*, pp. 1-12, Santa Clara, CA, June 2000.

[46] E. G. Coffman, P. Jelenković and P. Momčilović, "Provably Efficient Streaming Merging", *Proceedings 6th International Workshop on Web Caching and Content Distribution*, Boston, MA, June 2001.

[47] M. Crovella and A. Bestavros, "Self-similarity in World Wide Web traffic: Evidence and Possible Causes", *IEEE/ACM Transactions on Networking,* vol. 5, no. 6, pp. 835-846, December 1997.

[48] C. Cunha, A. Bestavros and M. Crovella, "Characteristics of WWW Client-Based Traces", Technical Report TR-95-010, Boston University Computer Science Department, June 1995.

[49] Y. K. Dalal and R. M. Metcalfe, "Reverse Path Forwarding of Broadcast Packets", *Communications of ACM*, vol. 21, no. 12, pp. 1040-1048, December 1978.

[50] A. Dan, D. Dias, R. Mukherjee, D. Sitaram, and R. Tewari, "Buffering and Caching in Large Scale Multimedia Servers", *Proceedings 40th IEEE Computer Conference (COMPCON): Technologies for the Information Superhighway*, pp 217-224, San Francisco, CA, March 1995.

[51] A. Dan and D. Sitaram, "Buffer Management Policy for an On-Demand Video Server", IBM Research Report RC 19347, T.J. Watson Research Center, Yorktown Heights, NY, January 1993.

[52] A. Dan and D. Sitaram, "Scheduling Policies for an On-Demand Video Server with Batching", *Proceedings ACM Multimedia*, San Francisco, CA, October 1994.

[53] A. Dan and D. Sitaram, "A Generalized Interval Caching Policy for Mixed Interactive and Long Video Environments", *Proceedings Multimedia Computing and Networking Conference (MMCN)*, San Jose, CA, January 1996.

[54] A. Dan and D. Sitaram, "Multimedia Caching Strategies for Heterogeneous Application and Server Environments", *Multimedia Tools and Applications*, vol. 4, pp. 279-312, May 1997.

[55] A. Dan, D. Sitaram and P. Shahabuddin, "Scheduling Policies for an On-demand Video Server with Batching", *Proceedings 2nd ACM International Multimedia Conference*, pp. 15-23, San Francisco, CA, October 1994.

[56] M. Dodge, "An Atlas of CyberSpaces",
http://www.geog.ucl.ac.uk/casa/martin/atlas/more_isp_maps.html

[57] D. L. Eager, M. C. Ferris and M. K. Vernon, "Optimized Regional Caching for On-Demand Data Delivery", *Proceedings Multimedia Computing and Networking (MMCN)*, San Jose, CA, January 1999.

[58] D. L. Eager, M. C. Ferris and M. K. Vernon, "Optimized Caching in Systems with Heterogeneous Client Populations", *Performance Evaluation, Special Issue on Internet Performance Modeling*, pp. 163-185, September 2000.

[59] D. L. Eager and M. K. Vernon, "Dynamic Skyscraper Broadcasts for Video-on-Demand", *Proceedings 4th International Workshop on Multimedia Information Systems (MIS)*, pp. 18-32, Istanbul, Turkey, September 1998.

[60] D. L. Eager, M. K. Vernon and J. Zahorjan, "Optimal and Efficient Merging Schedules for Video-on-Demand Servers", *Proceedings 7th ACM International Multimedia Conference*, Orlando, FL. November 1999.

[61] D. L. Eager, M. K. Vernon and J. Zahorjan, "Bandwidth Skimming: A Technique for Cost-Effective Video-on-Demand", *Proceedings Multimedia Computing and Networking (MMCN)*, San Jose, CA, January 2000.

[62] D. L. Eager, M. K. Vernon and J. Zahorjan, "Minimizing Bandwidth Requirements for On-Demand Data Delivery", *IEEE Transactions On Knowledge and Data Engineering, Special Section of invited papers from MIS'99*, vol. 13, no. 5, pp. 742-757, September/October. 2001.

[63] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma and L. Wei, "Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification", Request for Comments 2362, USC, Cisco, UMICH, Xerox, UCL, LBL, June 1998.

[64] "eTeach: Learning on Demand", http://eteach.engr.wisc.edu/newEteach/home.html

[65] M. Faloutsos, P. Faloutsos and C. Faloutsos, “On Power-Law Relationships of the Internet Topology”, *Proceedings ACM SIGCOMM*, Cambridge, MA, August 1999.

[66] L. Fan, P. Cao, J. Almeida and A. Broder, “Summary Cache: A Scalable Wide-Area Cache Sharing Protocol”, *IEEE/ACM Transactions on Networking*, pp. 281-293, vol. 8, no. 3, June 2000.

[67] Z. Fei, M. Ammar, E. Zegura, “Multicast Server Selection: Problems, Complexity and Solutions", *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 7, pp. 1399-1413, September 2002.

[68] L. Gao, “On Inferring Autonomous System Relationships on the Internet”, *Proceedings IEEE Global Internet Symposium*, San Francisco, CA, November 2000.

[69] L. Gao, Z.-L. Zhang, and D. Towsley, “Catching and Selective Catching: Efficient Latency Reduction Techniques for Delivering Continuous Multimedia Streams”, *Proceedings 7th ACM International Multimedia Conference*, pp 203-206, Orlando, FL. November 1999.

[70] M.R. Garey and D. S. Johnson, “Computer and Intractability: A Guide to the Theory of NP-Completeness”, W H Freeman, November 1979.

[71] Z. Ge, P. Jin, P. Shenoy, “A Demand Adaptive and Locality Aware (DALA) Streaming Media Server Cluster Architecture”, *Proceedings ACM 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, Miami Beach, FL, May 2002.

[72] Z. Ge, D. R. Figueiredo, S. Jaiswal and L. Gao, “On the Hierarchical Structure on the Logical Internet Graph”, *Proceedings SPIE International Symposium on Convergence of Information Technologies and Communication (ITCom)*, Denver, CO. August 2001.

[73] L. Golubchik, J. C. S. Lui, T. F. Tung, L. H. Chow, W. J. Lee, G. Franceschinis, C. Anglano, "Multi-Path Continuous Media Streaming: What are the Benefits?", *Proceedings IFIP WG 7.3 22nd International Symposium on Computer Performance Modeling and Evaluation*, Rome, Italy, September 2002.

[74] K. Guo, M. Buddhikot, Y. Chae and S. Suri, "RCache: Design and Analysis of Scalable, Fault Tolerant Multimedia Stream Caching Schemes", *Proceedings SPIE Conference on Scalability and Traffic Control in IP Networks*, Denver, CO, August 2001.

[75] M. Guo, M. Ammar, E. Zegura, "Selecting among Replicated Batching Video-on-Demand Servers", *Proceedings ACM 12 th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, Miami Beach, FL, May 2002.

[76] Y. Guo, S. Sen and D. Towsley, "Prefix Caching Assisted Periodic Broadcast: Framework and Techniques to Support Streaming for Popular Videos", *Proceedings IEEE International Conference on Communications (ICC)*, New York, NY, April/May 2002.

[77] J. Gwertzman and M. Seltzer, "The Case for Geographical Push-Caching", *Proceedings 5th Workshop on Hot Topics in Operating Systems*, pp. 51-55, Orcas Island, WA, May 1995.

[78] N. Harel, V. Vellanki, A. Chervenak, G. Abowd, U. Ramachandran, "Workload of a Media-Enhanced Classroom Server", *Proceedings IEEE 2nd Annual Workshop on Workload Characterization*, Austin, TX, October 1999.

[79] L. He, J. Grudin, and A. Gupta, "Designing Presentations for On-Demand Viewing", *Proceedings ACM Conference on Computer Supported Cooperative Work*, Philadelphia, PA, December 2000.

[80] M. Hofmann, T.S. E. Ng, K. Guo, S. Paul and H. Zhang, “Caching Techniques for Streaming Multimedia over the Internet”, Bell Labs Technical Report, April 1999.

[81] H. W. Holbrook and D. R. Cheriton, “IP Multicast Channels: Express Support for Large-Scale Single-Source Applications", *Proceedings ACM SIGCOMM*, Cambridge, MA, August/September 1999.

[82] K. Hua, Y. Cai and S. Sheu, “Patching: A Multicast Technique for True Video-on-Demand Services”, *Proceedings ACM Multimedia*, pp. 191-200, Bristol, U.K., September 1998.

[83] K. A. Hua and S. Sheu, “Skyscraper Broadcasting: A New Broadcasting Scheme for Metropolitan Video-on-Demand Systems”, *Proceedings ACM SIGCOMM*, pp. 89-100, Cannes, France, September 1997.

[84] National Lab for Applied Network Research, ICP working group, Internet Cache Protocol, http://ircache.nlanr.net/Cache/ICP, 1998.

[85] S. Jamin, C. Jiu, A. Kurc, D. Raz and Y. Shavitt, “Constrained Mirror Placement on the Internet”, *Proceedings IEEE INFOCOM*, Anchorage, AK, April 2001.

[86] S. Jin and A. Bestavros, “Temporal Locality in Web Request Streams: Sources Characteristics and Caching Implications”, *Proceedings ACM Sigmetrics*, Santa Clara, CA, June 2000.

[87] S. Jin and A. Bestavros, “GISMO: A Generator of Internet Streaming Media Objects and Workloads”, *ACM SIGMETRICS Performance Evaluation Review*, vol. 29, no.3, November 2001.

[88] S. Jin and A. Bestavros, “Scalability of Multicast Delivery for Non-Sequential Streaming Access”, *Proceedings ACM Sigmetrics*, Marina Del Rey, CA, June 2002.

[89] C. Jin, Q Chen and S. Jamin "Inet: Internet Topology Generator", Technical Report CSE-TR-433-00, University of Michigan, EECS Dept, 2000, http://topology.eecs.umich.edu/inet .

[90] M Kamath, K. Ramamritham and D. Towsley, "Continuous Media Sharing in Multimedia Database Systems", *Proceedings International Conference on Database Systems for Advanced Applications*, Singapore, April 1995.

[91] J. Kangasharju, F. Hartanto, M. Reisslein and K. W. Ross, "Distributing Layered Encoded Video through Caches", *Proceedings IEEE INFOCOM*, pp. 1791-1800, Anchorage, AK, April 2001.

[92] M. Karlsson, C. Karamanolis and M. Mahalingam, "A Framework for Evaluating Replica Placement Algorithms", Technical Report HPL-2002-219, HP Laboratories, August 2002.

[93] P. Krishnan, D. Raz and Y. Shavitt, "The Cache Location Problem", *IEEE/ACM Transactions on Networking*, vol. 8, no. 5, pp.568-582, October 2000.

[94] E. D. Lazowska, J. Zahorjan, G. S. Graham, K. C. Sevcik, Quantitative System Performance, Prentice Hall, 1984.

[95] S.-J. Lee, W-Y. Ma and B. Shen, "An Interactive Video Delivery and Caching System Using Video Summarization", *Proceedings 6th International Workshop on Web Caching and Content Distribution (WCW)*, pp. 309-326, Boston, MA, June 2001.

[96] B. Li, M. Golin, G.Italiano and X Deng, "The Optimal Placement of Web Proxies in the Internet", *Proceedings IEEE INFOCOM*, New York, NY, March 1999.

[97] D. Loguinov and H. Radha, "Measurement Study of Low-bitrate Internet Video Streaming", *Proceedings ACM SIGCOMM Internet Measurement Workshop (IMW)*, San Francisco, CA, November 2001.

[98] P. Lorenzettti, L. Rizzo and L.Vicisano, "Replacement Policies for a Proxy Cache", Technical Report, Universita di Pisa, Italy, December 1996.

[99] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, D. A. Spielman and V. Stelmann, "Practical Loss-Resilient Codes", *Proceedings 29th Annual ACM Symposium on Theory of Computing (STOC)*, El Paso, TX, May 1997.

[100] A. Mahanti, D. Eager and C. Williamson, "Temporal Locality and its Impact on Web Proxy Cache Performance", *Performance Evaluation, special issue on Internet Performance Modeling*, September 2000.

[101] A. Mahanti, D. L. Eager, M. K. Vernon, and D. Sundaram-Stukel, "Scalable On-Demand Media Streaming with Packet Loss Recovery", *Proceedings ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pp. 97-108, San Diego, CA, August 2001.

[102] A. Medina, A. Lakhina, I. Matta and J. Byers, "BRITE: An Approach to Universal Topology Generation", *Proceedings International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS)*, Cincinnati, Ohio, August 2001.

[103] A. Mena and J. Heidemann, "An Empirical Study of Real Audio Traffic", *Proceedings IEEE INFOCOM*, pp. 101-110, Tel Aviv, Israel, March 2000.

[104] J. Merwe, S. Sen and C. Kalmanek, "Streaming Video Traffic: Characterization and Network Impact", *Proceedings 7th International Workshop on Web Content Caching and Distribution (WCW)*, Boulder, CO, August 2002.

[105] Z. Miao and A. Ortega, "Proxy Caching for Efficient Video Services Over the Internet", *Proceedings Packet Video Workshop,* Columbia University, NY, April 1999.

[106] E. O'Neil, P. O'Neil and G. Weikum, "The LRU-k Page Replacement Algorithm for Database Disk Buffering", *Proceedings International Conference on Management of Data and Symposium on Principles of Database Systems*, pp. 297-306, Washington, DC, May 1993.

[107] B. Ozden, R. Rastogi and A. Silberschatz, "Buffer Replacement Algorithms for Multimedia Storage Systems", *Proceedings International Conference on Multimedia Computing and Systems,* Hiroshima, Japan, June 1996.

[108] B. Ozden, R. Rastogi and A. Silberschatz, "Disk Striping in Video Server Environments", *Proceedings International Conference on Multimedia Computing and Systems,* Hiroshima, Japan, June 1996.

[109] J. Padhye and J. Kurose, "An Empirical Study of Client Interactions with a Continuous-Media Courseware Server", *Proceedings 8th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV),*Cambridge, UK, July 1998.

[110] V. S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, E. M. Nahum, "Locality-Aware Request Distribution in Cluster-based Network Servers", *Proceedings 8th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VIII),* pp. 205-216, San Jose, CA, October 1998.

[111] J.-F. Pâris, S. W. Carter and D. D. E. Long, "A Hybrid Broadcasting Protocol for Video on Demand", *Proceedings Multimedia Computing and Networking (MMCN)*, San Jose, CA, January 1999.

[112] V. Paxson and S. Floyd, "Wide-Area Traffic: The Failure of Poisson Modeling", *IEEE/ACM Transactions on Networking*, vol. 3, no. 3, pp.226-244, June 1995.

[113] D. Patterson, "Computers for the Post-PC Era", *Sun Computer Systems SE Symposium*, Burlingame, CA, February 2000.

[114] D. Patterson and K. Keeton, "Hardware Technology Trends and Database Opportunities", *Sigmod 98 Keynote Address*, Seattle, WA, June 1998.

[115] L. Qiu, V. Padmanabhan and G. Voelker, "On the Placement of Web Server Replicas", *Proceedings IEEE INFOCOM*, Anchorage, AK, April 2001.

[116] P. Radoslavov, R. Govindam and D. Estrin, "Topology-Informed Internet Replica Placement", *Computer Communications*, vol. 25, no.4, pp. 394-392, March 2002.

[117] N. Race, D. Waddington and D. Shepherd, "An Experimental Dynamic RAM Video Cache", *Proceedings 10th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, Chapel Hill, NC, June 2000.

[118] S. Ramesh, I. Rhee and K. Guo, "Multicast with Cache (Mcache): An Adaptive Zero-Delay Video-on-Demand Service", *Proceedings IEEE INFOCOM*, Anchorage, AK, April 2001.

[119] "RealServer Administration Guide – RealSystem G2", RealNetworks, Inc., November 1998, http://docs.real.com/docs/serveradminguideg2.pdf .

[120] M. Reisslein, F. Hartanto and K. W. Ross, “Interactive Video Streaming with Proxy Servers", *Proceedings International Workshop on Intelligent Multimedia Computing and Networking (IMMCN)*, Atlantic City, NJ, February 2000.

[121] R. Rejaie and J. Kangasharju, “Mocha: A Quality Adaptive Multimedia Proxy Cache for Internet Streaming”, *Proceedings 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, Port Jefferson, NY, June, 2001.

[122] R. Rejaie, M. Handley, H. Yu and D. Estrin, “Proxy Caching Mechanism for Multimedia Playback Streams in the Internet”, *Proceedings 4th International WWW Caching Workshop*, San Diego, CA, March 1999.

[123] R. Rejaie, H. Yu, M. Handley and D. Estrin, “Multimedia Proxy Caching Mechanism for Quality Adaptive Streaming Applications in the Internet”, *Proceedings IEEE INFOCOM*, Tel Aviv, Israel, March 2000.

[124] Y. Rekhter and T. Li, “A Border Gateway Protocol 4 (BGP-4)”, Request for Comments 1771, J. Watson Research Center, IBM Corporation, Cisco Systems, March 1995, http://www.ietf.org/rfc/rfc1771.txt .

[125] Routing Information Service, http://www.ripe.net .

[126] Route Views Project, http://www.routeviews.org .

[127] J. R. Santos, R. Muntz and B. Ribeiro-Neto, “Comparing Random Data Allocation and Data Striping in Multimedia Servers”, *Proceedings ACM Sigmetrics*, Santa Clara, CA, June 2000.

[128] S. Sen, J. Rexford and D. Towsley, “Proxy Prefix Caching for Multimedia Streams”, *Proceedings IEEE INFOCOM*, New York City, NY, March 1999.

[129] S. Sen, L. Gao, J. Rexford and D. Towsley, "Optimal Patching Schemes for Efficient Multimedia Streaming", *Proceedings 9th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, Basking Ridge, N.J., June 1999.

[130] A. Silberschatz, J. Peterson and P. Galvin, *Operating System Concepts*, Addison Wesley, 1992.

[131] "The Skitter Project", http://www.caida.org/tools/measurement/skitter/ .

[131] J. Spirn, "Distance String Models for Program Behavior", *IEEE Computer*, vol. 9, no. 11, pp. 14-20, November, 1976.

[133] L. Subramanian, S. Agarwal, J. Rexford, R. Katz, "Characterizing the Internet Hierarchy from Multiple Vantage Points", *Proceedings INFOCOM*, New York, NY, June 2002.

[134] H. Tan, D. L. Eager, M. K. Vernon, and H. Guo, "Quality of Service Evaluations of Multicast Streaming Protocols", *Proceedings ACM Sigmetrics*, Marina del Rey, CA, June 2002.

[135] H. Tan, D. L. Eager, and M. K. Vernon, "Delimiting the Effectiveness of Scalable Streaming Protocols", *Proceedings IFIP WG 7.3 22nd International Symposium on Computer Performance Modeling and Evaluation (Performance)*, Rome, Italy, September 2002.

[136] W. Tavanapong, M. Tran, J. Zhou and S. Krishnamohan, "Video Caching Network for On-Demand Video Streaming", *Proceedings IEEE GLOBECOM*, Taipei, Taiwan, November 2002.

[137] R. Tewari, H. Vin, A. Dan, and D. Sitaram, "Caching in Bandwidth and Space Constrained Hierarchical Hyper-media Servers", Technical Report CS-TR-96-30, Department of Computer Sciences, University of Texas at Austin, January 1997

[138] R. Tewari, H. Vin, A. Dan and D. Sitaram, "Resource-Based Caching for Web Servers", *Proceedings SPIE/ACM Conference on Multimedia Computing and Networking (MMCN)*, San Jose, CA, January 1998.

[139] R. Tewari, "Architectures and Algorithms for Scalable Wide-Area Information Systems", PhD thesis, University of Texas at Austin, August 1998.

[140] Traceroute.org, "Public Route Server and Looking Glass List", http://www.traceroute.org .

[141] D. A. Tran, K. A. Hua and S. Sheu, "A New Caching Architecture for Efficient Video Services on the Internet", *Proceedings IEEE Symposium on Applications and the Internet (SAINT)*, Nara City, JAPAN, January/February 2002.

[142] E. Veloso, V. Almeida, W. Meira Jr., A. Bestavros, S. Jin, "A Hierarchical Characterization of a Live Streaming Media Workload", *Proceedings ACM SIGCOMM Internet Measurement Workshop*, Marseille, France, November, 2002.

[143] C. Venkatramani, O. Verscheure, P. Frossard and K-W. Lee, "Optimal Proxy Management for Multimedia Streaming in Content Distribution Networks", *Proceedings 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, Miami Beach, FL, May 2002.

[144] O. Verscheure, C. Venkatramani, P. Frossard and L. Amini, "Joint Server Scheduling and Proxy Caching for Video Delivery", *Proceedings 6th International Workshop on Web Caching and Content Distribution (WCW)*, Boston, MA, June 2001.

[145] M. Vishwanath and P. Chou, "An Efficient Algorithm for Hierarchical Compression of Video", *Proceedings IEEE International Conference on Image Processing*, Austin, Texas, November 1994.

[146] “Volera Website”, http://www.volera.com/ .

[147] B. Wang, S. Sen, M. Adler and D. Towsley, “Optimal Proxy Cache Allocation for Efficient Streaming Media Distribution”, *Proceedings IEEE INFOCOM*, New York, NY, June 2002.

[148] Y. Wang, Z.-L. Zhang, D. Du and D. Su, “A Network Conscious Approach to End-to-End Video Delivery over Wide Area Networks Using Proxy Servers”, *Proceedings IEEE INFOCOM*, San Francisco, CA, March 1998.

[149] S. Williams, M. Abrams, C. R. Standbridge, G. Abdulla and E. A. Fox, “Removal Policies in Network Caches for World-Wide Web Documents”, *Proceedings ACM SIGCOMM*, Stanford University, CA, August 1996.

[150] W. Willinger, M. Taqqu, R. Sherman and D. Wilson, “Self-Similarity Through High-Variability: Statistical Analysis of Ethernet LAN Traffic at the Source Level”, *Proceedings ACM SIGCOMM*, Cambridge, MA, Aug. 1995.

[151] “Windows Media Services SDK, Version 4.1”, Microsoft Corporation, http://msdn.microsoft.com/workshop/imedia/windowsmedia/sdk/wmsdk.asp .

[152] A. Wolman, G. Voelker, N. Sharma, N. Cardwell, M. Brown, T. Landray, D. Pinnel, A. Karlin and H. Levy, “Organization-Based Analysis of Web-Object Sharing and Caching”, *Proceedings 2nd Usenix Symposium on Internet Technology and Systems (USITS)*, Boulder, CO, October 1999.

[153] R. Wooster and M. Abrams, “Proxy Caching the Estimates Page Load Delays”, *Proceedings 6th International World Wide Web Conference*, Santa Clara, CA, April 1997.

[154] E. Zegura, K. Calvert and M. Donahoo, “A Quantitative Comparison of Graph-Based Model for Internet Topology”, *ACM/IEEE Transactions on Networking*, vol. 5, no. 6, pp. 770-783, December 1997.

[155] Y. Zhao, D. L. Eager, and M. K. Vernon, “Network Bandwidth Requirements for Scalable On-Demand Streaming”, *Proceedings IEEE INFOCOM*, New York, NY, June 2002.

[156] G. K. Zipf, Human Behavior and the Principal of Least-Effort, Addison-Wesley, Cambridge, MA 1949.

Appendix A

Further Media Workload Data

This appendix provides additional data for the workload characteristics analyzed in chapter 4.

Figure 56 shows the cumulative distribution of the stack distances for accesses to each one-minute file segment in eTeach for different days. The higher the number of small stack distances, the stronger the temporal locality. Temporal locality seems to vary significantly across different days on the server. Given the roughly uniform access observed for the blocks of the most popular files, a weaker temporal locality may be due to a greater change in the file popularity during the period analyzed. For instance, on October 10^{th}, which was the day before the course first exam, temporal locality is weak possibly due to students reviewing all the previous lectures for the exam.

Table 22 shows the typical values of the parameters of the two Zipf-like distributions used to model file access frequency in both eTeach and BIBS servers. These parameters include, for each distribution, the number of files, the total access probability and the Zipf parameter θ. Table 23 provides a summary of the accesses to infrequently requested files in eTeach. As discussed in section 4.3.4, like in the BIBS server, a large fraction of the new files is accessed only once, and most of those files are not accessed again within the next 8 hours.

Figure 57 shows the distribution of the number of accesses to ten-second segments for files with different relative file access frequency for two other days in eTeach. As discussed in chapter 4, the distributions is roughly uniform for the most popular files and skewed towards earlier

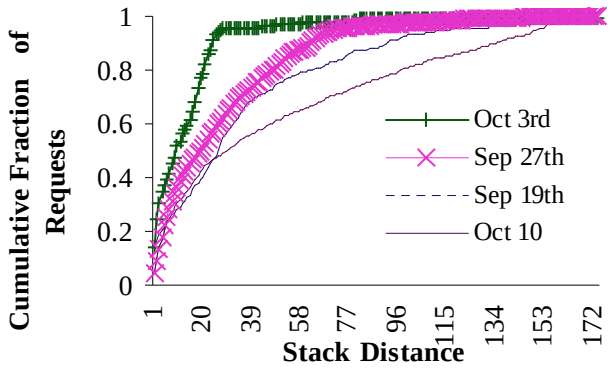


Figure 56: Temporal Locality for File Segment Access in eTeach

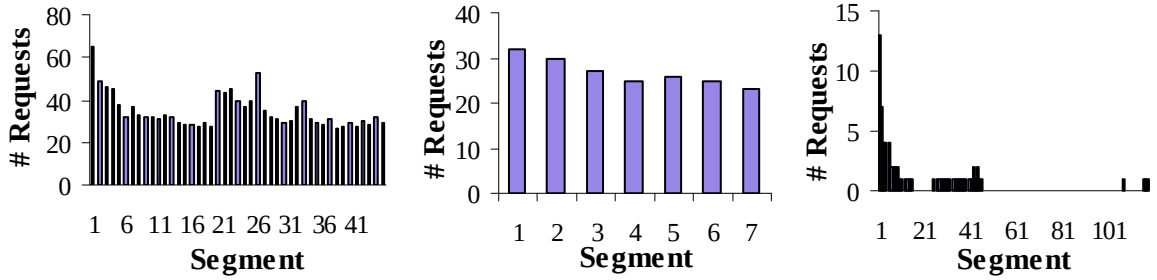
Table 22: Typical Parameter Values for the Distribution of File Access Frequencies

File Size (mins)	Server	Day	First Zipf Distribution			Second Zipf Distribution		
			Total Prob.	# files	θ	Total Prob.	# files	θ
0–5	eTeach	9/13–14, 9/19–20, 10/3	0.7–0.85	2	2–2.5	0.15–0.3	3–12	1–1.5
		9/21, 10/4	0.6	2	1.7	0.4–0.45	9-17	0.8
		9/26–27	0.84	2	0.9–1.25	0.16	7–11	2
		10/10–11	0.8	7	0.2–0.25	0.2	7	1.2
	BIBS	2/20, 3/23	0.1–0.2	2	0	0.8–0.9	56–85	0.45–0.55
		5/8–9, 5/14	0.2	2-4	0.3-0.33	0.8–0.85	30–60	0.35-0.38
5–10	eTeach	9/27	0.8	2	0.3625	0.2	2	1.6
		10/09	0.85	2	0.5208	0.15	2	10.05
10-15	eTeach	9/19	0.89	2	3.6	0.11	2	0
		9/20	0.97	2	5.4	0.03	2	2.4
50–55	BIBS	2/21–22	0.5–0.6	6–9	0.4–0.45	0.4–0.5	58–72	0.6–0.65
		5/8–11	0.5–0.6	12–18	0.2–0.35	0.4–0.5	110–130	0.45–0.55

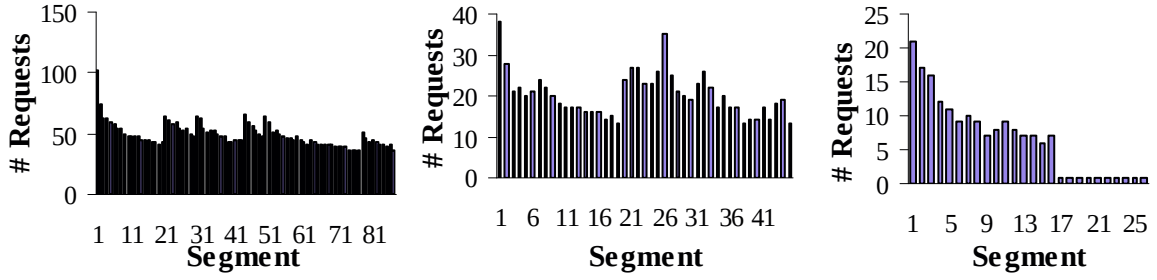
segments for less popular files. Figure 58 shows the hourly evolution of file popularity for two other days for both BIBS and eTeach. The amount of fluctuations depend on the day analyzed but it is more significant on the days closer to course exams, i.e., March 23rd in BIBS, and October 3rd, in eTeach.

Table 23: Summary of Accesses to New Media Files in eTeach

Day	h	Median # new	Average # new accessed once / # new	Time Until Next Access			
				% > 4 hr	% > 8 hr	% > 16 hr	% > 32 hr
9/15	2	1	0.57	67	67	67	60
	8	1	0.56	69	69	69	69
9/27	2	2	0.52	41	41	36	18
	8	2	0.33	64	64	55	36
10/3	2	1	0.54	67	67	56	33
	8	1	0.70	67	67	56	33
10/10	2	3	0.40	46	31	31	31
	8	4	0.50	56	33	33	33

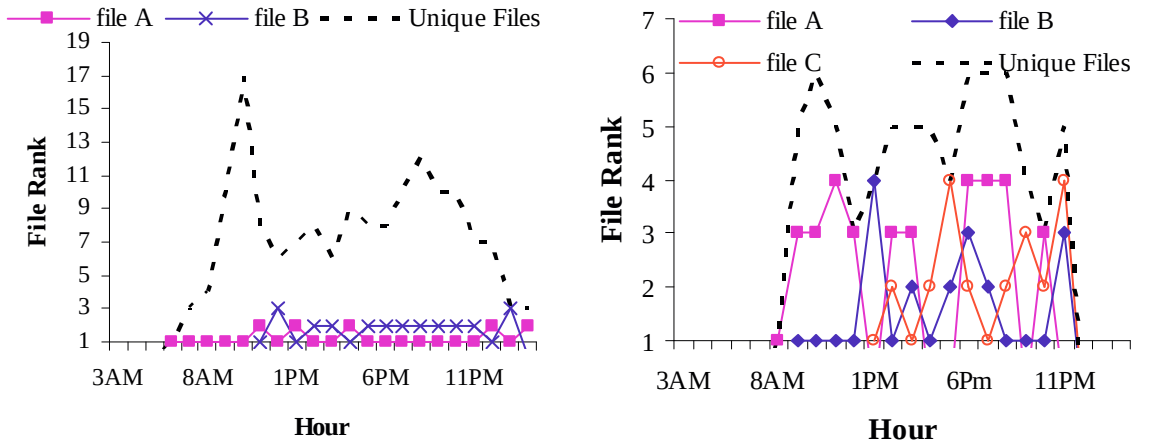


a) Most Accessed File b) 3rd Most Accessed File (October 3rd) c) 5th Most Accessed



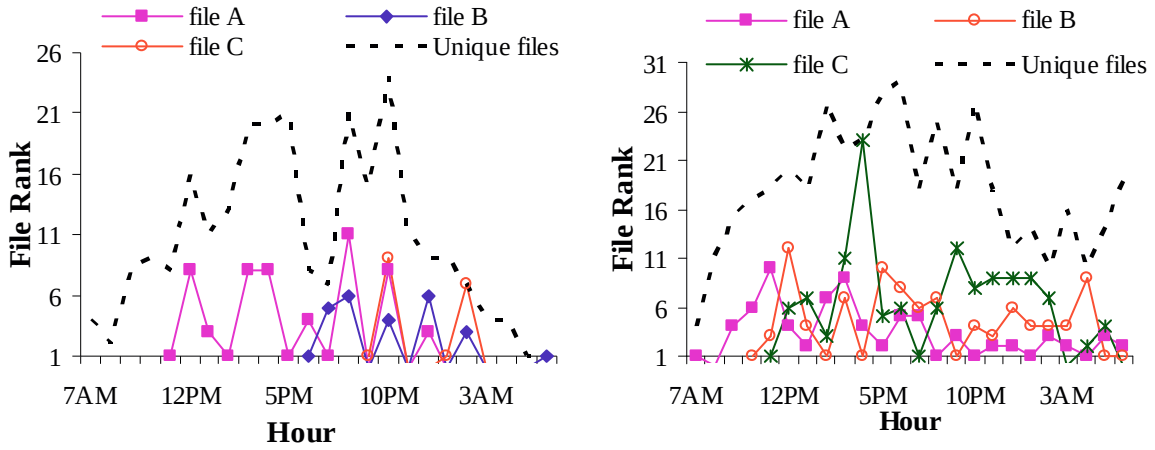
a) Most Accessed File b) 4th Most Accessed File (October 10th) c) 18th Most Accessed File

Figure 57: Further Example Distributions of Accesses to 10-Second File Segments in eTeach



a) eTeach, September 27th

b) eTeach, October 3rd



c) BIBS, February 2nd

d) BIBS, March 23rd

Figure 58: Further Examples of Hourly Evolution of File Popularity

Appendix B

Required Origin Server Bandwidth for BWSkim+Batch(b)

This appendix derives the formula used, in section 6.1.5, for $B_{origin,last\_segment}(f)$, the average required origin server bandwidth to deliver the last segment of length $1 - 2f$ of the file for the BWSkim+Batch(b) protocol. Recall that the arrivals at the origin server for the last segment are approximately deterministic, with rate $N^*_{origin}(f)$. In the following, we derive the formula for the required server bandwidth for bandwidth skimming to deliver an object, of length equal $T = 1$, assuming deterministic arrivals, at rate N per T. The formula is applied to determine $B_{origin,last\_segment}(f)$ by replacing N with $N^*_{origin}(f)$.

We define the members of a merging tree to be the clients that end up merged together with an earlier full stream. A full stream does not merge with any previous stream. We define a merging tree as all the (pieces of) streams that are delivered to those clients. Figure 59, illustrates the merges that occur for a sequence of arrivals for an object with arrival rate $N = 6$ per object duration. The x-axis represents time and the y-axis represents the position on the file for each stream, as the stream progresses. To simplify, we show only the successful merges. The client requests are grouped into merging trees and one is shown separately. In the tree, leaves, shown as gray nodes, represent client requests, internal nodes represent merging points, branches of the tree represent (pieces of)

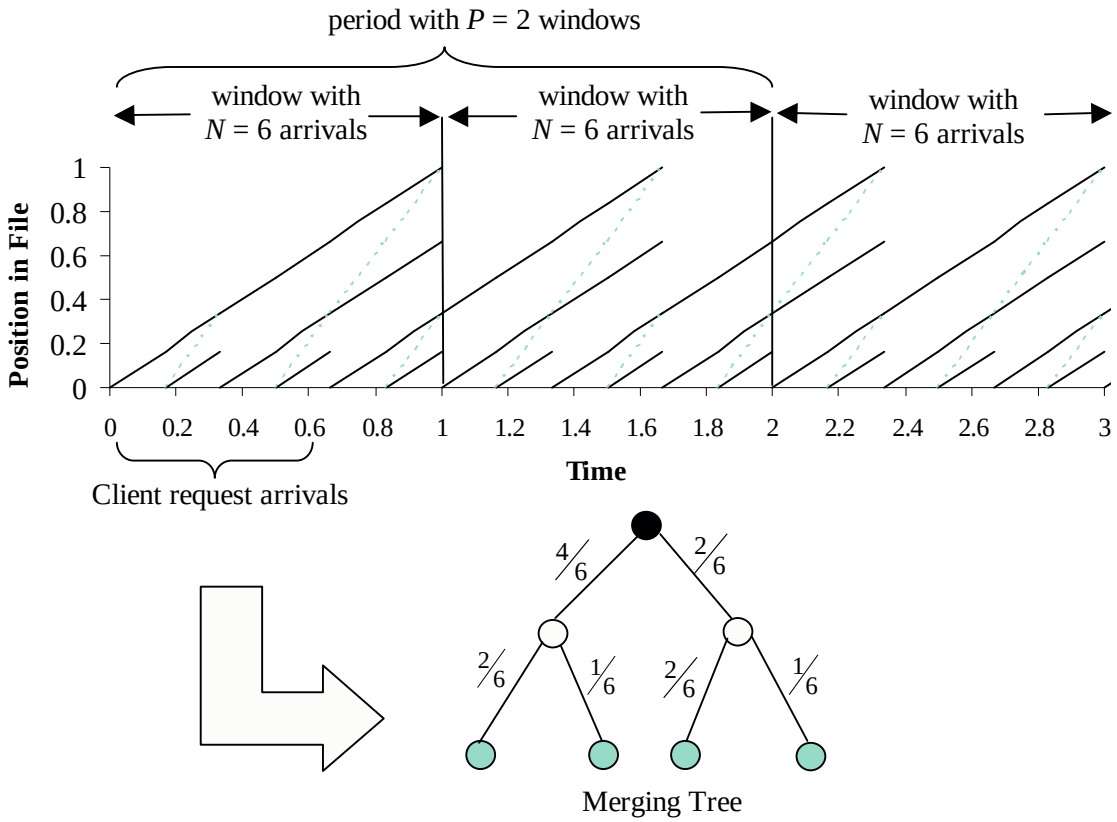


Figure 59: Operation of Bandwidth Skimming for Deterministic Request Arrivals

streams and the root represents the end of the full stream. The branches of the tree are labeled with the length (in units of streams) of the piece.

We divide time into windows, where each window contains *N* arrivals. Furthermore, we group windows into periods, where a period is defined as the number of windows it takes for the pattern of (pieces of) streams per window to repeat. In Figure 59, a period consists of 2 windows. To calculate the average bandwidth, we sum up the lengths of the pieces of the streams that fit in a tree, multiply it by the number of trees per period and then divide it by the number of arrivals during a period.

The number of trees per period depends on the number of clients in each tree and on the duration of the period. To determine the number of clients in each tree, note that length of the earliest (full) stream in a tree can be calculated by summing up the length of the pieces between two consecutive merges of subsequent streams with the full stream. Furthermore, the lengths of these pieces increase at rate two, as new merges occur. In other words, defining a piece unit to be equal to $1/N$, the length of the first piece (before the first merge) is two. After the 2nd stream merges with the full stream, a new piece of length four is added. Then, after the 3rd (and 4th) streams merge with it, a new piece of length eight is added and so on, until the stream is terminated as it reaches the end of the file. Since the length of the stream is equal to N pieces, the following inequality must be true:

$$N \geq \sum_{i=1}^{k} 2^i = 2^{k+1} - 2 \quad \Rightarrow \quad N + 2 \geq 2^{k+1}$$

$$k = \lfloor \log_2 (N+2) \rfloor - 1 \tag{14}$$

In (14), k is the height of the merging tree and 2^k is the number of clients that end up merged with the full stream, i.e., the number of clients in the tree. By building each tree from bottom to top, one can see there is a pattern on the length of each branch of the tree (i.e., each stream), defined by how the merges occur. This pattern is clear in the tree in Figure 59. In each subtree, the left branch has twice the length of the right branch, because the right stream merged into the left stream. Furthermore, the number of streams drops to half as one visits each level in the tree towards the root. Thus, the required bandwidth per tree, in units of (full) streams, B_{tree}, can be calculated by summing the bandwidth required for each level i of the tree, and adding to that sum the bandwidth corresponding the last piece of the file (B_{last}) that is delivered after all merges occur (in Figure 59, that piece has length 0). Given k, the height of the tree, computed as in (14), B_{tree} is calculated as:

$$B_{tree} = \sum_{i=1}^{k} (\frac{2^k}{2^i}) \times (2^{i-1} + 2^i) + B_{last} = (\sum_{i=1}^{k} 2^{k-i} \times 2^{i-1} + \sum_{i=1}^{k} 2^{k-i} \times 2^i) + B_{last}$$

$$B_{tree} = k \times (2^{k-1} + 2^k) + B_{last}$$

$$B_{last} = N - \sum_{i=1}^{k} 2^i$$

$$B_{tree} = 3 \times k \times 2^{k-1} + N - 2^{k+1} + 2 \quad (15)$$

In order to calculate the number of trees that fit in a period, recall that each window has N arrivals and that each tree includes 2^k arrivals. Thus, given P, the duration of a period in number of windows, the number T of trees per period is given by:

$$T = \frac{P \times N}{2^k} \quad (16)$$

Given that the total bandwidth per period is $T \times B_{tree}$, the average required server bandwidth, B, is calculated by dividing this number by the total number of arrivals during a period, i.e., $P \times N$, as follows:

$$B(N) = \frac{T \times B_{tree}}{P \times N} = \frac{\frac{P \times N}{2^k} \times B_{tree}}{P \times N} = \frac{B_{tree}}{2^k} = \frac{3 \times k \times 2^{k-1} + N - 2^{k+1} + 2}{2^k} = \frac{3 \times k}{2} + \frac{N+2}{2^k} - 2 \quad (17)$$

Equations (14) and (17) correspond to the formulas for $k(f)$ and $B_{origin,last\_segment}(f)$, shown in section 6.1.5 with N replaced by $N^*_{origin}(f)$, the arrival rate for the last segment of the suffix at the origin server.

Appendix C

Extensions to the Cost Models for Scalable CDNs

This appendix provides extensions to the design models shown in chapter 6 to include non-uniform file sizes and delivery rates as well as heterogeneous client request rates and proxy capacities. The BWSkim(b) protocol is used to illustrate the required changes to the models. The models for BWSkim/U(b) and BWSkim+Batch[/U](b) can be modified in a similar way.

C.1 Non-Uniform File Sizes and Delivery Rates

Let T_i be the file duration of file i. Let R be the highest file delivery rate among all files in the system and let r_i be the delivery rate of file i, relative the R (i.e., $0 < r_i \leq 1$). The normalized arrival rate for each file i, N_i, is given by $N_i = \lambda T_i$. The required proxy and origin server bandwidth, B_{proxy} and B_{origin}, for each file i delivered by BWSkim(b), expressed in units of streams at rate R, are redefined as:

$$B_{proxy}(f_i, N_i, P, b) = r_i \, \eta \, \ln(1 + f_i \, N_{proxy,i} / \eta), \tag{18}$$

$$B_{origin}(f_i, N_i, P, b) = r_i \, \eta \, \ln(1 + N_{origin,i} / \eta), \tag{19}$$

where: $N_{proxy,,i} = N_i / P$ and

$$N_{origin,i} = \frac{\eta(1 - f_i) N_i}{f_i N_i / P + \eta}$$

As in chapter 6, the total required origin and proxy server bandwidth is obtained by summing the expressions for B_{proxy} and B_{origin} over all files.

C.2 Heterogeneous Client Request Rates and Proxy Capacities

In this section, we assume uniform file sizes and delivery rates, but the extensions described in the previous section can be easily added afterwards. We consider a system with P proxy servers, with one distinct proxy, with total request rate, disk bandwidth and storage capacity possibly different from the request rates and capacities of the other P-1 non-distinct proxies [58]. In that case, the fraction of each file cached at the distinct proxy may be different from the fraction cached at the non-distinct proxies.

Given T, the file duration, let N^d and N^{nd} be request rate at the distinct proxy and at *all* non-distinct proxies, respectively and let N be the overall request rate, i.e., $N = N^d + N^{nd}$. Similarly, f^d and f^{nd} are defined as the fraction of the file cached at the distinct and at each non-distinct proxy, respectively. Given that $N^d_{proxy} = N^d$ and $N^{nd}_{proxy} = N^{nd}/P\text{-}1$, B^d_{proxy} and B^{nd}_{proxy}, the required server bandwidth for the distinct and each non-distinct proxy, respectively, is defined, using equation (6) defined in chapter 6, as follows:

$$B^d_{proxy}(f^d, N^d, b) = \eta \ln(1 + f^d \, N^d_{proxy} / \eta) \tag{20}$$

$$B^{nd}_{proxy}(f^{nd}, N^{nd}, P, b) = \eta \ln(1 + f^{nd} \, N^{nd}_{proxy} / \eta), \tag{21}$$

In order to calculate B_{origin}, four different scenarios are identified: 1) the file is not cached at any proxy (i.e., $f^d = f^{nd} = 0$); 2) the file is (partially) cached *only* at the distinct proxy (i.e., $f^d > 0$, f^{nd}

= 0); 3) the file is (partially) cached *only* at the non-distinct proxies (i.e., $f^d = 0$, $f^{nd} > 0$) and 4) the file is (partially) cached at all proxies (i.e., $f^d > 0$, $f^{nd} > 0$).

If the file is not cached at any proxy, the arrival rate at the origin server is given by the total arrival rate $N = N^d + N^{nd}$ and B_{origin} can be calculated as:

$f^d = f^{nd} = 0$: $$B_{origin}(f^d, f^{nd}, N^d, N^{nd}, P, b) = \eta \ln(1 + N/\eta) \quad (22)$$

If the file is cached only at the distinct proxy, B_{origin} can be split into two components, $B_{origin,1}$, the required origin bandwidth to deliver the fraction f^d to the clients of the P-1 non-distinct proxies, and $B_{origin,2}$, the required origin bandwidth to deliver the suffix to the clients of all proxies. $B_{origin,1}$ and $B_{origin,2}$ are defined, using the basic equations developed in chapter 6, as:

$f^d > 0$, $f^{nd} = 0$:

$$B_{origin}(f^d, f^{nd}, N^d, N^{nd}, P, b) = B_{origin,1}(f^d, N^{nd}, P, b) + B_{origin,2}(f^d, N^d, N^{nd}, P, b), \quad (23)$$

$$B_{origin,1}(f^d, N^{nd}, P, b) = \eta \ln(1 + f^d N^{nd}/\eta),$$

$$B_{origin,2}(f^d, N^d, N^{nd}, P, b) = \eta \ln(1 + N_{origin}/\eta),$$

where N_{origin}, the rate at which "requests" for the suffix arrive at the origin from all proxies, is derived using equation (8), in chapter 6, for each type of proxy (distinct and non-distinct) :

$$N_{origin} = \frac{\eta(1 - f^d)N^d}{f^d N^d + \eta} + \frac{\eta(1 - f^d)N^{nd}}{f^d \times N^{nd}/(P-1) + \eta}. \quad (24)$$

Similarly, if the file is cached only at the non-distinct proxies, B_{origin} is split into $B_{origin,1}$, the required origin bandwidth to deliver the fraction f^{nd} to the clients of the distinct proxy, and $B_{origin,2}$,

the required bandwidth to deliver the suffix to the clients of all proxies, which are derived, similarly to equations (23)-(24), as follows:

$f^d = 0, f^{nd} > 0$:

$$B_{origin}(f^d, f^{nd}, N^d, N^{nd}, P, b) = B_{origin,1}(f^{nd}, N^d, P, b) + B_{origin,2}(f^{nd}, N^d, N^{nd}, P, b), \quad (25)$$

$$B_{origin,1}(f^{nd}, N^d, P, b) = \eta \ln(1 + f^{nd} N^d / \eta),$$

$$B_{origin,2}(f^d, N^d, N^{nd}, P, b) = \eta \ln(1 + N_{origin} / \eta),$$

$$N_{origin} = \frac{\eta(1 - f^{nd})N^d}{f^{nd} N^d + \eta} + \frac{\eta(1 - f^{nd})N^{nd}}{f^{nd} \times N^{nd} / (P-1) + \eta} .$$

The most complex scenario is when a fraction of the file is cached at all proxies. Two cases must be considered: $f^d \le f^{nd}$ and $f^{nd} \le f^d$. For each such case, B_{origin} is split into two components. One component, $B_{origin,1}$, corresponds to the required bandwidth to deliver the intermediate segment of length $| f^d - f^{nd} |$ that is not cached in at least one of the proxies. The other component, $B_{origin,2}$, corresponds to the required bandwidth to deliver the last suffix of the file to the clients of all proxies. $B_{origin,1}$ and $B_{origin,2}$, for each of the two cases, are derived using a reasoning similar to the one used for the derivation of each component of B_{origin} in the two previous scenarios.

$0 < f^d \le f^{nd}$:

$$B_{origin}(f^d, f^{nd}, N^d, N^{nd}, P, b) = B_{origin,1}(f^d, f^{nd}, N^d, P, b) + B_{origin,2}(f^{nd}, N^d, N^{nd}, P, b), \quad (26)$$

$$B_{origin,1}(f^d, f^{nd}, N^d, P, b) = \eta \times \left[\ln(1 + f^{nd} N^d / \eta) - \ln(1 + f^d N^d / \eta) \right],$$

$$B_{origin,2}(f^{nd}, N^{d}, N^{nd}, P, b) = \eta \ln(1 + {N_{origin}}/{\eta}),$$

$$N_{origin} = \frac{\eta(1 - f^{nd})N^{d}}{f^{nd}N^{d} + \eta} + \frac{\eta(1 - f^{nd})N^{nd}}{f^{nd} \times {N^{nd}}/{(P-1)} + \eta}.$$

$0 < f^{nd} \leq f^{d}$:

$$B_{origin}(f^{d}, f^{nd}, N^{d}, N^{nd}, P, b) = B_{origin,1}(f^{d}, f^{nd}, N^{d}, P, b) + B_{origin,2}(f^{nd}, N^{d}, N^{nd}, P, b), \quad (27)$$

$$B_{origin,1}(f^{d}, f^{nd}, N^{nd}, P, b) = \eta \log(1 + \frac{N_{origin,\text{int}ermediate\_segment}}{\eta}),$$

$$N_{origin,\text{int}ermediate\_segment} = \frac{\eta N^{nd} \times (f^{d} - f^{nd})}{f^{nd}\ {N^{nd}}/{P-1} + \eta},$$

$$B_{origin,2}(f^{nd}, N^{d}, N^{nd}, P, b) = \eta \ln(1 + {N_{origin}}/{\eta}),$$

$$N_{origin} = \frac{\eta(1 - f^{d})N^{d}}{f^{d}N^{d} + \eta} + \frac{\eta \times (1 - relative\_fraction) \times N_{nd,last\_segment}}{relative\_fraction \times N_{nd,last\_segment} + \eta},$$

$$N_{nd,last\_segment} = \frac{\eta N^{nd}(1 - f^{nd})}{f^{nd} \times \frac{N^{nd}}{P-1} + \eta},$$

$$relative\_fraction = \frac{f^{d} - f^{nd}}{(1 - f^{nd})}.$$

Appendix D

Examples of CDN Design Problems

Figures 60 and 61 show example of two delivery system design problem where client nodes, labeled with their symbol in Table 19 (chapter 7), are represented by lightly shaded nodes in the diagram. To simplify the diagram in Figure 60, all bi-directional links are shown as undirected arcs.

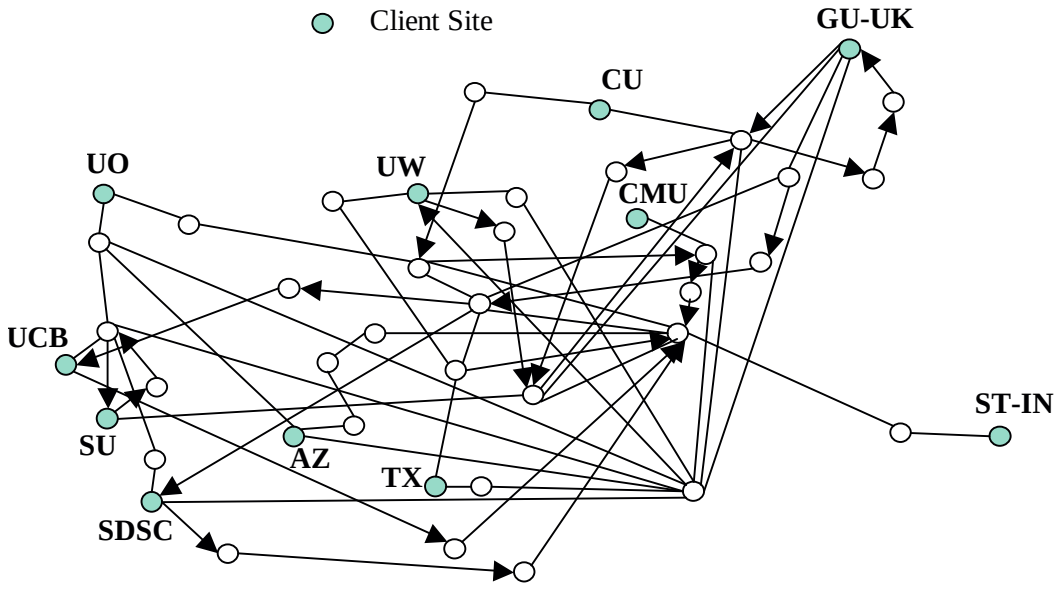


Figure 60: Delivery System Design Problem: Example A (AS level topology)

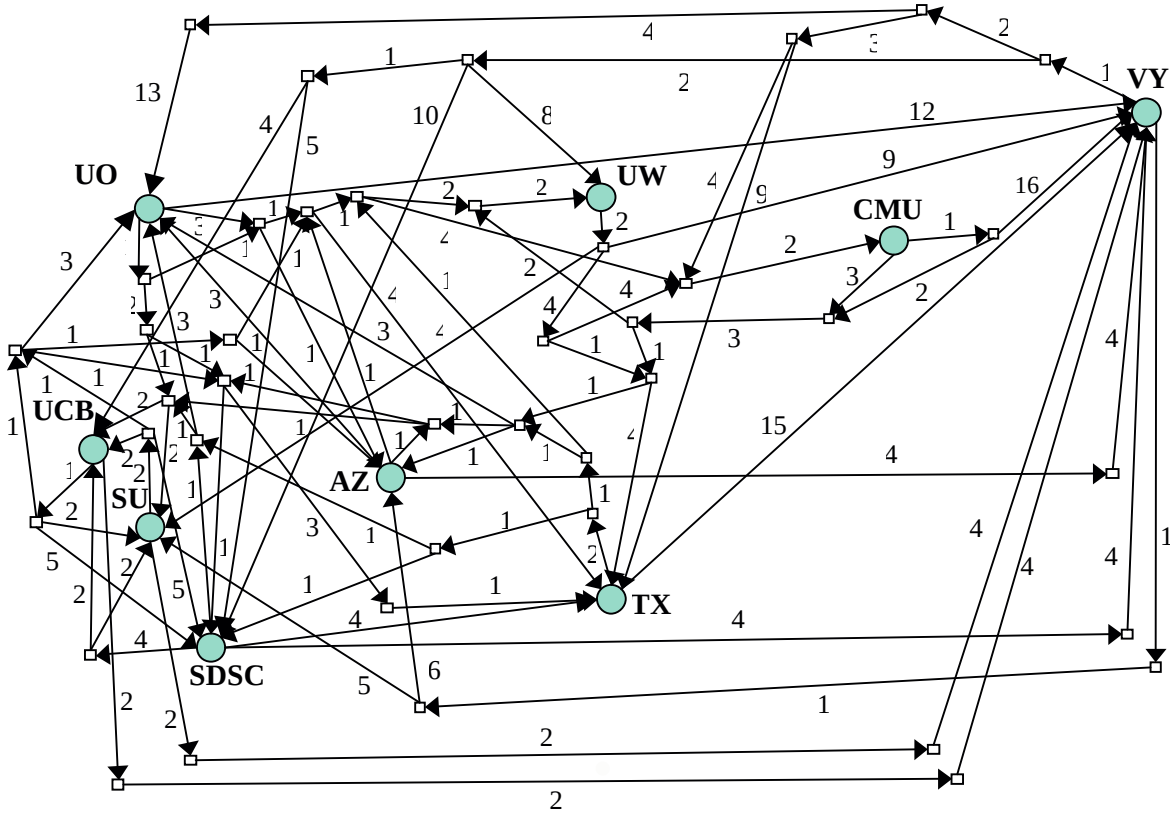


Figure 61: Delivery System Design Problem: Example B (router level topology)

Appendix E

Penalty from Using Alternative Routing and Placement Strategies for Scalable CDNs

E.1 Maximum Penalty from Using Alternative Routing Strategies

This appendix uses the simple topology shown in Figure 62, with one server *S* and two clients *A* and *B*, to analyze the efficiency of using one of two canonical delivery trees, namely the tree of shortest paths (t_{sp}) or the tree of fewest links (t_{fl}), for scalable delivery. We compare the network delivery cost of each such tree against the optimal, referred to as the network delivery cost of the tree of minimum cost, t_{min}.

Each tree t_x, $x \in \{sp, fl, min\}$, is defined by three parameters, shown in Figure 62, that determine the path segments: a_x, the length (in number of hops) of a shared path segment from the server *S* to an interior node I_x, and b_x and c_x, the lengths of two unshared segments from I_x to the clients. In the following, we will interchangeably refer to each segment and its length by the corresponding parameter. Let B_{sp}, B_{fl} and B_{min} be the network delivery cost of t_{sp}, t_{fl} and t_{min}, respectively. The following analysis derives the penalties, in terms of network delivery cost increase compared to the minimum, associated with t_{sp} or t_{fl} for arbitrary values of a_x, b_x and c_x, $x \in \{sp, fl, min\}$.

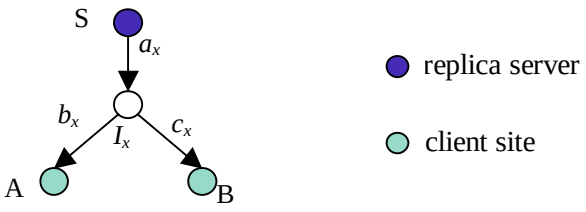


Figure 62: Canonical Delivery Tree with Two Client Sites($x \in \{sp, fl, min\}$)

Without loss of generality, we assume $N_A \leq N_B$, where N_A and N_B are the demands of clients *A* and *B*. The cost associated with t_{sp} or t_{fl} depend on the network bandwidth for each tree, defined as:

$$B_x = a_x \log(N_A + N_B + 1) + b_x \log(N_A + 1) + c_x \log(N_B + 1), \quad \text{for all } x \in \{sp, fl, min\}, \tag{28}$$

and the penalties are given by:

$$Penalty(t_{sp}) = \frac{B_{sp} - B_{\min}}{B_{\min}} = \frac{(a_{sp} - a_{\min})\log(N_A + N_B + 1) + (b_{sp} - b_{\min})\log(N_A + 1) + (c_{sp} - c_{\min})\log(N_B + 1)}{a_{\min}\log(N_A + N_B + 1) + b_{\min}\log(N_A + 1) + c_{\min}\log(N_B + 1)} \tag{29}$$

$$Penalty(t_{fl}) = \frac{B_{fl} - B_{\min}}{B_{\min}} = \frac{(a_{fl} - a_{\min})\log(N_A + N_B + 1) + (b_{fl} - b_{\min})\log(N_A + 1) + (c_{fl} - c_{\min})\log(N_B + 1)}{a_{\min}\log(N_A + N_B + 1) + b_{\min}\log(N_A + 1) + c_{\min}\log(N_B + 1)} \tag{30}$$

The *maximum* penalty associated with t_{sp} is derived by first defining constraints on the parameters a_{sp}, b_{sp}, and c_{sp}, for given a_{min}, b_{min} and c_{min}, such that both trees are valid (i.e., t_{sp} contains the shortest paths to the clients and t_{min} has the lowest network bandwidth). These constraints impose lower and upper bounds on the values that can be assigned to each parameter, which are used to derive an upper bound on expression (29). A similar approach is also used to derive the maximum penalty associated with t_{fl}. The following derivations will show that: a) the maximum penalty associated with t_{sp} is bounded and this bound can be expressed as a function of the lengths of the path segments in t_{sp} and the shortest distance between the two clients and b) the maximum penalty associated with t_{fl} is not bounded. We first derive the maximum penalty associated with t_{sp}.

The following constraints must hold so that the t_{sp} and t_{min} are valid:

(1) $B_{sp} \geq B_{min}$.

This constraint guarantees that t_{sp} does not have lower network bandwidth than t_{min}.

(2) $a_{sp} + b_{sp} \leq a_{min} + b_{min} \Rightarrow b_{sp} - b_{min} \leq a_{min} - a_{sp}$.

(3) $a_{sp} + c_{sp} \leq a_{min} + c_{min} \Rightarrow c_{sp} - c_{min} \leq a_{min} - a_{sp}$.

Constraints (2) and (3) guarantee that the paths in t_{sp} are the shortest paths to the clients.

(4) $a_{sp} \leq a_{min}$.

To prove that this constraint must hold, we show that if $a_{sp} > a_{min}$ then $B_{sp} < B_{min}$, i.e., t_{min} is not valid. Applying the upper-bounds on $b_{sp} - b_{min}$ and $c_{sp} - c_{min}$, derived from constraints (2) and (3) in equation (29), we have:

$$Penalty(t_{sp}) = \frac{B_{sp} - B_{\min}}{B_{\min}} = \frac{(a_{sp} - a_{\min})\log(N_A + N_B + 1) + (b_{sp} - b_{\min})\log(N_A + 1) + (c_{sp} - c_{\min})\log(N_B + 1)}{a_{\min}\log(N_A + N_B + 1) + b_{\min}\log(N_A + 1) + c_{\min}\log(N_B + 1)} <$$

$$< \frac{(a_{sp} - a_{\min})\log(N_A + N_B + 1) + (a_{\min} - a_{sp})\log(N_A + 1) + (a_{\min} - a_{sp})\log(N_B + 1)}{a_{\min}\log(N_A + N_B + 1) + b_{\min}\log(N_A + 1) + c_{\min}\log(N_B + 1)}$$

$$Penalty(t_{sp}) < \frac{(a_{sp} - a_{\min})[\log(N_A + N_B + 1) - \log(N_A + 1) - \log(N_B + 1)]}{a_{\min}\log(N_A + N_B + 1) + b_{\min}\log(N_A + 1) + c_{\min}\log(N_B + 1)} \quad (31)$$

Note that $\log(N_A + N_B + 1) < \log(N_A + 1) + \log(N_B + 1)$. Thus, if $a_{sp} > a_{min}$, the penalty associated with t_{sp} is negative, i.e., $B_{sp} < B_{min}$. Therefore, $a_{sp} \leq a_{min}$

The above constraints are used to derive an upper bound on expression (29) as follows. Starting from equation (31), reorganizing the numerator, and expressing $b_{min} + c_{min}$ as $f(a_{min} - a_{sp})$, an upper bound on the penalty associated with t_{sp} is given by:

$$Penalty(t_{sp}) < \frac{(a_{\min} - a_{sp})[\log(N_A + 1) + \log(N_B + 1) - \log(N_A + N_B + 1)]}{a_{\min}\log(N_A + N_B + 1) + b_{\min}\log(N_A + 1) + c_{\min}\log(N_B + 1)} <$$

$$\frac{(a_{\min} - a_{sp})\log(N_A + 1)}{a_{\min}\log(N_A + N_B + 1) + b_{\min}\log(N_A + 1) + c_{\min}\log(N_B + 1)} <$$

$$\frac{(a_{\min} - a_{sp})\log(N_A + 1)}{(a_{\min} + b_{\min} + c_{\min})\log(N_A + 1)} = \frac{(a_{\min} - a_{sp})}{(a_{\min} - a_{sp} + b_{\min} + c_{\min})} = \frac{1}{(f + 1)}$$

$$Penalty(t_{sp}) < \frac{1}{(f + 1)} < 100\% \tag{32}$$

In practice, given a server and two clients, one can use bounds on the values of a_{min}, b_{min} and c_{min} to calculate $f = (b_{min} + c_{min})/(a_{min} - a_{sp})$ and thus the upper-bound on the penalty of using t_{sp} as follows. The shortest distance d between the two clients can be used as a lower bound on $b_{min} + c_{min}$ and the sum $a_{sp} + b_{sp} + c_{sp}$ can be used as an upper bound on the value of a_{min}, i.e., $b_{sp} + c_{sp}$ is used as an upper-bound for $a_{min} - a_{sp}$. The value of f and the maximum penalty from using t_{sp} can then be defined as:

$$f = \frac{d}{b_{sp} + c_{sp}} \qquad MaxPenalty(t_{sp}) < \frac{1}{d/b_{sp} + c_{sp} + 1} = \frac{b_{sp} + c_{sp}}{d + b_{sp} + c_{sp}} \tag{33}$$

Therefore, given a server, two clients and the tree of shortest paths from the server to the two clients, the larger the shortest distance between the clients, relative to the sum of the lengths of the unshared paths, the closer to optimal is the network delivery cost of the tree of shortest paths[7].

A tighter bound on $a_{min} - a_{sp}$, and thus on the penalty of t_{sp} can be derived from constraint (1):

[7] We point out that in case there are multiple t_{sp}'s from a server to two clients, the one that has the longest shared segment is the one with lowest cost.

$$B_{min} \le B_{sp} \Rightarrow \quad a_{\min}\log(N_A+N_B+1)+b_{\min}\log(N_A+1)+c_{\min}\log(N_B+1) \le a_{sp}\log(N_A+N_B+1)+b_{sp}\log(N_A+1)+c_{sp}\log(N_B+1)$$

$$a_{\min}-a_{sp} \le \frac{(b_{sp}-b_{\min})\log(N_A+1)+(c_{sp}-c_{\min})\log(N_B+1)}{\log(N_A+N_B+1)} < \frac{b_{sp}\log(N_A+1)+c_{sp}\log(N_B+1)}{\log(N_A+N_B+1)} \quad (34)$$

In chapter 7, to simplify the explanation, we will use the less tight upper bound on the penalty defined in (33) However, in practice, expression (34) can be used to more accurately estimate the value of f and the penalty associated with t_{sp}.

The maximum penalty associated with t_{fl} is derived next, by first identifying the following constraints as necessary so that t_{fl} and t_{min} are valid:

(1) $a_{fl} + b_{fl} + c_{fl} \le a_{min} + b_{min} + c_{min} \Rightarrow a_{fl} - a_{min} \le b_{min} - b_{fl} + c_{min} - c_{fl}$

This constraint guarantees that the number of links in t_{fl} is not larger than the number of links in t_{min}.

(2) $a_{fl}\log(N_A+N_B+1) + b_{fl}\log(N_A+1) + c_{fl}\log(N_B+1) \ge$

$a_{min}\log(N_A+N_B+1) + b_{min}\log(N_A+1) + c_{min}\log(N_B+1)$

This constraint guarantees that B_{fl}, the network bandwidth of t_{fl}, is no lower than B_{min}.

(3) $a_{min} \le a_{fl}$.

To prove that this constraint must be true we show that if $a_{min} > a_{fl}$, a third tree that has lower network delivery cost than the given t_{min} can be built, i.e., t_{min} is not valid. First, we show that if $a_{min} > a_{fl}$ then $a_{fl} + c_{fl} > a_{min} + c_{min}$. If the path segments that have the higher network bandwidth in

t_{fl} (i.e., a_{fl} and c_{fl}) have total length shorter than the total length of the corresponding path segments in t_{min}, then, given constraint (1), clearly, $B_{fl} < B_{min}$. Thus, if $a_{min} > a_{fl}$, then $a_{fl} + c_{fl} > a_{min} + c_{min}$.

However, if $a_{fl} + c_{fl} > a_{min} + c_{min}$, then $b_{fl} < b_{min}$, otherwise constraint (1) does not hold. But if a) $a_{min} > a_{fl}$, b) $a_{fl} + c_{fl} > a_{min} + c_{min}$ and c) $b_{fl} < b_{min}$ then a third tree with path segments a_{fl}, b_{fl} and $a_{min} - a_{fl} + c_{min}$ can be built and this tree has network bandwidth, B_{new}, lower than B_{min}:

$$
\begin{aligned}
B_{new} &= a_{fl}\log(N_A+N_B+1)+b_{fl}\log(N_A+1)+(a_{\min}-a_{fl}+c_{\min})\log(N_B+1)= \\
&= a_{fl}(\log(N_A+N_B+1)-\log(N_B+1))+b_{fl}\log(N_A+1)+(a_{\min}+c_{\min})\log(N_B+1)< \\
&< a_{\min}(\log(N_A+N_B+1)-\log(N_B+1))+a_{\min}\log(N_B+1)+b_{fl}\log(N_A+1)+c_{\min}\log(N_B+1)= \\
&= a_{\min}\log(N_A+N_B+1)+b_{fl}\log(N_A+1)+c_{\min}\log(N_B+1)< \\
&< a_{\min}\log(N_A+N_B+1)+b_{\min}\log(N_A+1)+c_{\min}\log(N_B+1)=B_{\min}
\end{aligned}
$$

An upper bound on the maximum penalty associated with t_{fl} is derived using the above constraints as follows. Applying the upper bound on $a_{fl} - a_{min}$, in constraint (1) to expression (30):

$$
\begin{aligned}
Penalty(t_{fl}) &= \frac{(a_{fl}-a_{\min})\log(N_A+N_B+1)+(b_{fl}-b_{\min})\log(N_A+1)+(c_{fl}-c_{\min})\log(N_B+1)}{a_{\min}\log(N_A+N_B+1)+b_{\min}\log(N_A+1)+c_{\min}\log(N_B+1)} \le \\
&\le \frac{(b_{\min}-b_{fl}+c_{\min}-c_{fl})\log(N_A+N_B+1)+(b_{fl}-b_{\min})\log(N_A+1)+(c_{fl}-c_{\min})\log(N_B+1)}{a_{\min}\log(N_A+N_B+1)+b_{\min}\log(N_A+1)+c_{\min}\log(N_B+1)} \\
&= \frac{(b_{\min}-b_{fl})[\log(N_A+N_B+1)-\log(N_A+1)]+(c_{fl}-c_{\min})[\log(N_A+N_B+1)-\log(N_B+1)]}{a_{\min}\log(N_A+N_B+1)+b_{\min}\log(N_A+1)+c_{\min}\log(N_B+1)} < \\
&< \frac{(b_{\min}-b_{fl})[\log(N_A+N_B+1)-\log(N_A+1)]}{a_{\min}\log(N_A+N_B+1)+b_{\min}\log(N_A+1)+c_{\min}\log(N_B+1)}
\end{aligned}
$$

$$
Penalty(t_{fl}) < \frac{b_{\min}[\log(N_A+N_B+1)-\log(N_A+1)]}{a_{\min}\log(N_A+N_B+1)+b_{\min}\log(N_A+1)+c_{\min}\log(N_B+1)} \qquad (35)
$$

We could not find a simpler expression for the upper bound on the penalty associated with t_{fl}. Note that it can be arbitrarily high (if $N_B >> N_A$), depending on the values of a_{min}, b_{min}, c_{min}, N_A and N_B. Taking the limit of expression (35), the maximum penalty converges to $b_{min}/(a_{min}+c_{min})$ as N_B

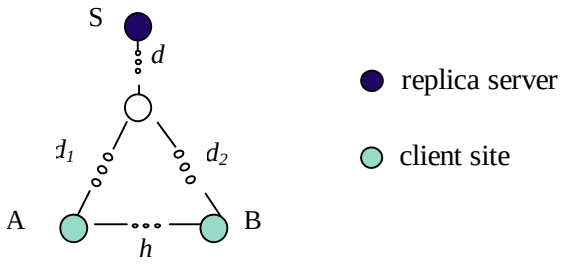


Figure 63: Topology for Analyzing Placement of a Second Replica

increases. Compared to the fixed upper bound for t_{sp}, t_{fl} may be significantly more expensive than t_{sp} if the paths to the clients are much longer, despite the sharing.

E.2 Maximum Penalty from Placement Optimized for Unicast Delivery

Consider the topology shown in Figure 63, with one replica already placed at the top node and two clients, *A* and *B*. The shortest distances between any two nodes in the topology are shown in the figure. As discussed in section 7.4, assuming the placement of a second replica is constrained to one of the two clients, the optimal solution for conventional and scalable delivery may be different if the client with lowest demand is farther away from all other nodes. This section derives the maximum penalty, in terms of network delivery cost increase, if the optimal solution for conventional delivery is used for a scalable system. Note that this analysis assumes that the second replica is placed at one of the two clients. In general, the optimal placement might be at another (interior) node, not shown in Figure 63. However, we make this simplifying assumption in order to be able to derive insights into the maximum penalty associated with conventional placement strategies, which are useful for more general topologies.

Without loss of generality, we assume $N_A \leq N_B$. Client *A*, with lowest demand N_A, is farther away from the other nodes if either $d_2 + d \leq d_1 + d \leq h$ or $d_2 + d \leq h \leq d_1 + d$. For the following analysis, we assume $d_2 + d \leq d_1 + d \leq h$, although a similar result is obtained for the other case.

Let $B_{conventional}(i)$ and $B_{scalable}(i)$ be the bandwidth cost *after* the second replica is placed at client node $i \in \{A, B\}$, for conventional unicast and scalable multicast delivery, respectively. The optimal solutions for the placement of the second replica for conventional and scalable delivery are different if $B_{scalable}(A) < B_{scalable}(B)$ and $B_{conventional}(B) < B_{conventional}(A)$, i.e.:

$$(d_2 + d)\log(N_B + 1) < (d_1 + d)\log(N_A + 1) \quad and \quad (d_1 + d)N_A < (d_2 + d)N_B$$

or, more concisely, $$\frac{\log(N_B + 1)}{\log(N_A + 1)} < \frac{d_1 + d}{d_2 + d} < \frac{N_B}{N_A} \tag{36}$$

The maximum cost penalty from using the optimal placement for conventional delivery in this configuration for a scalable CDN can be derived by assuming $(d_1+d)/(d_2+d)$ is as large as possible, without violating (36), i.e., $(d_1+d)/(d_2+d) < N_B/N_A$. The maximum penalty is then given by:

$$\begin{aligned} Max \quad Penalty(unicast) &= \frac{B_{scalable}(B) - B_{scalable}(A)}{B_{scalable}(A)} = \frac{(d_1 + d)\log(N_A + 1) - (d_2 + d)\log(N_B + 1)}{(d_2 + d)\log(N_B + 1)} \\ &= \frac{(d_1 + d)\big/(d_2 + d)\log(N_A + 1) - \log(N_B + 1)}{\log(N_B + 1)} < \frac{N_B}{N_A} \times \frac{\log(N_A + 1)}{\log(N_B + 1)} - 1 \end{aligned} \tag{37}$$

Fixing N_A, the maximum penalty is unbounded, i.e., it increases with N_B, provided (36) holds.