Scalable Download Protocols

A Thesis Submitted to the College of

Graduate Studies and Research

In Partial Fulfillment of the Requirements

For the Degree of Doctor of Philosophy

In the Department of Computer Science

University of Saskatchewan

Saskatoon, Saskatchewan

By

Niklas Carlsson

Permission to Use

Abstract

Scalable on-demand content delivery systems, designed to effectively handle increasing request rates, typically use service aggregation or content replication techniques. Service aggregation relies on one-to-many communication techniques, such as multicast, to efficiently deliver content from a single sender to multiple receivers. With replication, multiple geographically distributed replicas of the service or content share the load of processing client requests and enable delivery from a nearby server.

Previous scalable protocols for downloading large, popular files from a single server include batching and cyclic multicast. Analytic lower bounds developed in this thesis show that neither of these protocols consistently yields performance close to optimal. New hybrid protocols are proposed that achieve within 20% of the optimal delay in homogeneous systems, as well as within 25% of the optimal maximum client delay in all heterogeneous scenarios considered.

In systems utilizing both service aggregation and replication, well-designed policies determining which replica serves each request must balance the objectives of achieving high locality of service, and high efficiency of service aggregation. By comparing classes of policies, using both analysis and simulations, this thesis shows that there are significant performance advantages in using current system state information (rather than only proximities and average loads) and in deferring selection decisions when possible. Most of these performance gains can be achieved using only "local" (rather than global) request information.

Finally, this thesis proposes adaptations of already proposed peer-assisted download techniques to support a streaming (rather than download) service, enabling playback to begin well before the entire media file is received. These protocols split each file into pieces, which can be downloaded from multiple sources, including other clients downloading the same file. Using simulations, a candidate protocol is presented and evaluated. The protocol includes both a piece selection technique that effectively mediates the conflict between achieving high piece diversity and the in-order requirements of media file playback, as well as a simple on-line rule for deciding when playback can safely commence.

Acknowledgements

This work has been carried out under the supervision of Derek Eager. Throughout my time in department he has been an excellent teacher, mentor, and friend. Learning from him has been extremely rewarding and is an experience which I will always treasure. My research has also greatly benefited from feedback and support provided by Dr. Mary Vernon.

My advisory committee, consisting of Winfried Grassmann, Mark Keil, Ralph Deters, and Chris Soteros, has provided thoughtful comments and feedback, making every meeting a great learning experience. Thank you to my external examiner Dr. Janelle Harms, who traveled from Edmonton to attend my defence.

I would also like to thank all the graduate students, staff, and faculty who have made the atmosphere within the Department a comfortable place to work. In particular, I would like to thank Jan Thompson for making life easier on graduate students; Gord McCalla for making sure that I have human contact even at odd hours; Dwight Makaroff for his willingness to talk hockey; and finally, Anirban Mahanti for providing me with guidance through life as a graduate student and beyond.

I would like to thank my parents Olle and Lena who have provided me with more help and better guidance than I could have ever asked for, my sister Malin, as well as my grandparents. Your support and love made this degree worth a lot more.

A special thank you goes to my partner Alexa Briggs. In addition to supporting me in my pursuit of this degree, she has been a great teacher of the English language, my best friend, and a great source of joy and laughter.

Finally, I would like to send a thought to my very special four legged friend Bobby Lue. Although you are no longer with us, you have a big part in this thesis. She was a great listener and would always keep me company when I needed it the most.

Thank you all for your support!

Table of Contents

List of Tables

List of Figures

List of Common Notation

This table summarizes the most commonly used notation in this thesis.

Symbol	**Page**	**Definition**
A	40, 107	Average client delay
B	40, 81	Average rate at which service cost is incurred (equal to the average server bandwidth)
b	40, 119	Maximum sustainable client reception rate (equal to download bandwidth capacity of a client/peer)
b_j^m	119	Maximum sustainable download rate for any of *j*'s download connections
C	81	Total service delivery cost
c	81	Access cost per unit of service received for all client groups *i* and replicas $j,\ i \neq j$
D	40, 81	Maximum client delay
f	41, 108	Batching delay parameter (fraction of times *n*+1 should be used as a threshold, rather than *n*)
L	40, 80, 122	Service required by each requesting client (equal to the file size)
N	80	Number of replicas
n	41, 108	Batching delay parameter (threshold value on the number of requests)
q_i	81	Average fraction of its service that a group *i* client receives from other than replica *i*
P, p	–	Probabilities
r	40, 80	Transmission rate on a multicast channel
r_{ij}	119	Transfer rate on a connection from peer *i* to peer *j*
T, t	–	Time
u_i	119	Upload bandwidth capacity of peer *i*
u_i^m	119	Maximum sustainable upload rate for any of *i*'s upload connections
α	90, 121	Zipf parameter
Δ	40	Batching delay parameter (threshold value on the maximum time until service)
δ_{ij}	119	1 if *i* is transmitting to *j* (i.e., *j* is *interested* in *i* and has been *unchoked* by *i*); 0 otherwise
$\partial_{f>0}$	63, 109	1 or 0, based on logic comparison
φ	122	Rate at which peers defect from the system before having completed download
η	127	Exponential decay factor
λ	40, 80, 122	File request rate

List of Acronyms

This table summarizes acronyms used in this thesis.

Acronym	Page	Definition
AS	14	Autonomous System
batching/cbd	40	Batching/constant batching delay
batching/rbd	41	Batching/request-based delay
BGP	15	Boarder Gateway Protocol
CBT	15	Core-Based Trees
CDN	3	Content Distribution Network
cyclic/cd,l	55	Cyclic/constant delay, listeners
cyclic/cd,bot	57	Cyclic/constant delay, bounded on-time
cyclic/l	42	Cyclic/listeners
cyclic/rbd,l	60	Cyclic/request-based delay, listeners
cyclic/rbd,cot	61	Cyclic/request-based delay, controlled on-time
cyclic/w2,l	48	Cyclic/wait for second, listeners
DNS	6	Domain Name System
DVMRP	15	Distance Vector Multicast Routing Protocol
EWMA	130	Expected weighted moving average
FCFS	23	First Come First Serve
hlb	71	Heterogeneous lower bound
IGMP	14	Internet Group Membership Protocol
ISP	4	Internet Service Providers
KBR	19	Key-based routing
LTA	130	Long term average
LWF	23	Longest Wait First
MBGP	15	Multiprotocol Boarder Gateway Protocol
MOSPF	15	Multicast Open Shortest Path First
MRF	23	Most Request First
MRFL	23	Most Request First Lowest
MSDP	16	Multicast Source Discovery Protocol
NRU	20	Normalized Resource Usage
PIM-SM	15	Protocol Independent Multicast Sparse Mode
RP	15	Rendezvous point
sa	50	Shifted arrivals
slp	46	Send as late as possible
s-cyclic/l	73	Shared cyclic/listeners

Glossary

Batching protocol
A type of service aggregation protocol, in which clients having requested a file, wait to begin receiving the file until the beginning of a multicast (or broadcast) transmission, which collectively serves a set of waiting clients.

BitTorrent
A peer-assisted download protocol, in which a file is split into smaller pieces that can be downloaded in parallel from different peers.

Bulk data
Data or files for which there is no advantageous order in which data should be retrieved.

Choke algorithm
Algorithm used, by BitTorrent, to determine which peers to upload (and not to upload) pieces of a file to.

Client reception rate
The rate at which data is received by the client.

Content Distribution Network (CDN)
Interconnected servers distributed across the network, which allows the content to be effectively replicated, clients to be served by nearby replicas, and the content distributor to maintain control over the content.

Continuous media files
Media files, such as audio and video, that continuously must be rendered at specified rates.

Cyclic multicast protocol
A type of service aggregation protocol, in which the file data is cyclically transmitted on a multicast channel, which clients begin listening to at an arbitrary point in time, and continue listening to until all of the file data has been received.

Digital fountain
A cyclic multicast protocol, in which the file data is erasure coded such that a client listening to the channel can recreate the original content after having retrieved an arbitrary set of data equal (or slightly larger) in size as the original file.

Download bandwidth capacity
The maximum sustainable rate at which data can be received by the client.

Download protocol
Protocol used to transfer bulk data to clients. The main metric of these protocols is the time until the entire file is fully downloaded.

Erasure coding
Coding technique used to accommodate packet losses.

Leecher
BitTorrent peer which does not have a complete copy of a file, and currently is downloading pieces of the file.

Multicast

Family of techniques used to set up forwarding trees and to forward the content (through these distribution trees), from one or more source to multiple receivers.

Multicast channel

The server and network resources used to deliver each multicast to each member of a multicast group.

Multicast group

A collection of nodes interested in receiving the same multicast transmission.

Upload bandwidth capacity

The maximum sustainable rate at which data can be transferred by the client.

Peer-assisted protocol

Protocol in which peers contribute to the collective power of the system by making (part of) their resources available.

Peer-to-peer system

Systems consisting of peers.

Piece selection policy

Policy used by BitTorrent peer to determine the next piece to request for download.

Proxy caches

Content caches located at servers embedded between clients and the origin server, which intercept client requests and (in the case they have a stored copy) serves them on behalf of the origin server.

Poisson arrival process

A memoryless arrival process with constant arrival rate, or equivalently, an arrival process with inter-arrival times that are independent and exponentially distributed.

Replica

A server which has a copy of the replicated file (or service).

Replica selection policy

Policy used to determine which replica should serve a given client request.

Seeder

A BitTorrent peer which has a complete copy of a file (hence not requiring additional data to be downloaded), yet uploading pieces of the file to other peers.

Server bandwidth

The rate at which data is transferred by a server.

Service aggregation technique

Technique that allows multiple client requests to be served together in a manner that is more efficient than individual service.

Streaming protocol

File transfer protocol for continuous media files that allows playback to begin before a file is completely retrieved.

Tit-for-tat policy

Policy used by BitTorrent, giving upload preference to peers that provide the highest download rates.

Chapter 1
Introduction

With tremendous improvements in network bandwidth and computer capabilities many new high-bandwidth applications have emerged in the entertainment, business, and scientific communities. In contrast to traditional content distribution systems, such as TV and radio channels, many of these new applications operate on an on-demand basis and only serve clients when explicit requests for service are made.

As on-demand applications are becoming more popular, content providers are faced with the problem of distributing enormous amounts of data to a growing population of client requests. For example, the size of a full length movie may be on the order of gigabytes. On-demand dissemination of such files to many different clients, potentially widely distributed across the Internet, requires significant server and network resources. Therefore, the rate at which a system can serve client requests is often limited by the server (and/or network) bandwidth available for dissemination, where bandwidth refers to the amount of data that can be transferred per time unit by the server (and/or across some network connection).

Two basic service models commonly used for on-demand delivery of stored data are *download* and *streaming*. With download, clients download the entire file before making use of it. In this context the main performance metric is the time until the entire file is downloaded. Streaming, on the other hand, utilizes the in-order playback characteristics of media files, such as video, to allow playback to begin well before all of the file data is retrieved. To increase the likelihood that each part of the media file is retrieved before its playback time, streaming techniques generally require that some initial portion of the file is retrieved, and stored into a buffer, before starting playback. Maintaining a buffer of file data is especially important in environments with widely varying (playback and/or retrieval) rates. With streaming, the primary metric of interest is the startup delay until playback can safely begin.

For content delivery systems to handle high request rates, it is important that protocols are designed such that either the resource requirements increase sub-linearly with increasing request rate, or the resources available for content delivery increase linearly with request rate. Using scalable techniques can allow a content distributor with limited resources to provide its customers with better service, handle a higher request rate, and/or reduce its resource requirements (and hence also its delivery costs). Throughout this thesis the scalability and resource requirements of different delivery protocols and architectures are considered. Of particular interest is the best achievable delivery service, given some available resources.

The remainder of this chapter is organized as follows. Section 1.1 provides an overview of existing scalable content delivery approaches. Section 1.2 defines the objectives of the thesis. The primary contributions are outlined in Section 1.3. Section 1.4 gives the organization of the remainder of the thesis.

1.1 Scalable Content Delivery

Before discussing scalable delivery architectures and protocols, consider the limitations of the basic client-server model, in which a content provider hosts all its content at a single server, and client requests are served individually. Such systems, independently of the scheduling algorithm used, require resource usage directly proportional to the number of requests. With limited resources this can easily result in unbounded client delays or dropped requests. Further, both the server itself and the network connectivity to the server will be potential bottlenecks and act as single points of failure.

Scalability can be achieved using service *aggregation* (e.g., [2, 6, 10, 12, 26, 31, 57, 146, 150, 176, 180, 182]) or *replication* (e.g., [93, 94, 96, 133]) techniques. With aggregation, multiple client requests for the same file are collectively served. These techniques often rely on one-to-many delivery multicast techniques, which build efficient dissemination trees from a single sender to multiple receivers. With replication, multiple geographically distributed replicas of the content share the load of processing client requests, offload the origin content server, and enable delivery from nearby replica servers. For example, replicas may be proactively pushed out to replica servers across a Content Distribution Network (CDN) or reactively replicated at proxy caches in response to client requests. Replication is also utilized in peer-assisted systems, in which other clients having obtained, or are currently obtaining, some content are willing to serve as additional replica servers.

1.1.1 Service Aggregation

Rather than serving each request individually, service aggregation techniques attempt to serve multiple requests simultaneously, in a manner that is more efficient than individual service. These techniques often utilize multicast, in which the server can use a single send operation to deliver content to all of the requesting clients. Multicast service employs a multicast delivery tree to disseminate the content. This tree can be constructed either by network routers (e.g., [178, 123, 19, 18, 60, 61, 87, 138]) or by application-level software (e.g., [45, 42, 95, 193, 153, 39, 139, 40]). When serving multiple requests simultaneously, multicast can significantly decrease the bandwidth requirements at the server, as well as the total bandwidth required throughout the network.

Multicast-based service aggregation techniques have been proposed both in the context of download and streaming. Previous scalable protocols for downloading large, popular files from a single server include batching [66, 182] and cyclic multicast [146, 10, 26, 176, 31, 150, 27]. With batching, clients wait to begin receiving a requested file until the beginning of its next multicast transmission, which collectively serves all of the waiting clients that have accumulated up to that point. With cyclic multicast, the file data is continually being multicast. Clients can begin listening to the multicast at an arbitrary point in time, and continue listening until all of the file data has been received.

In the context of streaming, scalable service aggregation protocols include periodic broadcast protocols [5, 177, 90, 119, 88] and immediate service protocols [79, 36, 89, 67, 68, 69, 76]. To allow playback to begin quickly, with immediate service protocols, a new stream is started for each client request, delivering the beginning of the file. To allow later clients to catch up with earlier clients, with respect to the portion of the file that has been received, clients may also listen to earlier streams. At the point a stream is no longer needed (since the clients listening to it have already received the data it is delivering, by listening to earlier streams) it can be terminated. With periodic broadcast protocols, segments of files are periodically multicast according to some schedule. Clients are provided with a schedule for listening to the various multicasts that ensures that all data is received in time.

To accommodate packet losses many service aggregation techniques, including some cyclic multicast and periodic broadcast protocols, utilize erasure codes [148, 31, 164, 121]. With erasure coding, a file of size N blocks is encoded into M blocks ($M > N$) such that reception of only a subset of the M blocks (of total number N or slightly larger) is sufficient to allow recreation of the file. For example, a client listening to a cyclic multicast can recover from a packet loss by continuing to listen until a sufficient amount of erasure-coded data has been received [146, 31]. Erasure codes also simplify content delivery in systems utilizing replication [30]. A client is able to download erasure coded blocks from multiple servers with minimal duplicate block receptions, as long as $M >> N$. The ratio M/N is called the stretch factor of the coding scheme.

1.1.2 Replication

The first and simplest replication approach is for the *content distributor* to invest in a *server farm* consisting of a set of mirror servers among which an intelligent content switch can direct requests. However, without service aggregation this approach requires server and network resources to scale linearly with the number of requests and may result in a single point of failure if all replicas are located in the same sub-network or behind a common network bottleneck. Three alternative replication strategies are (i) *proxy caching*, (ii) *Content Distribution Networks* (*CDNs*), and (iii) *peer-to-peer networks*.

1.1.2.1 Proxy Caching

Internet Service Providers (ISPs) or user communities (such as businesses or universities) often embed proxy servers or organization level caches at the boundary of their networks. By redirecting client requests through a proxy server, that caches previously requested files, these architectures allow requests to be served by a nearby proxy server, rather than the origin content source. Specifically, if the proxy has a cached copy of the requested content, the proxy can serve the request itself, otherwise the proxy first retrieves a copy from the server. Proxy caching can reduce network bandwidth usage as well as improve the perceived client performance.

To determine which files a proxy cache should retain copies of, various replacement policies have been developed [147, 1, 34]. Such policies may exploit (i) the highly variable popularities of web objects, (ii) short-term temporal locality in the object request stream, whereby an object may experience several closely spaced requests, and (iii) the correlations among requests for different objects. However, with relatively cheap storage, disk caches can be made large enough to make replacement policies less pertinent. The effectiveness of proxy caching is largely determined by the proportion of cacheable objects, and the rate these objects are updated, in comparison with the request rate of each object [181]. To increase the probability that a copy of the requested file can be found close to the requesting client, many systems have been proposed that use cooperative caching, whereby proxy caches close to each other cooperate in serving client requests [71, 181].

Common for all caching techniques is that they work best if the content is static; however, as data is pulled from the origin server and stored at individual proxy caches the content provider loses control over the content and can not provide service guarantees. Therefore, the origin server may include a directive with data that it sends to the proxy cache, requesting a short maximum cache lifetime, forcing caches to refresh their content relatively frequently, and thus reducing their effectiveness.

1.1.2.2 Content Distribution Networks

Content Distribution Networks (CDNs) [62, 166] are provisioned by content brokers. By distributing servers across the network and interconnecting them at the application-level the content broker can create a distributed overlay infrastructure, which it can use to provide content distribution services for content providers. Selling their services to content providers, these networks are generally designed to provide attractive services such as reliable and high quality delivery to the content provider's customers. These systems relieve content providers from investing in infrastructure and offload the origin content servers. With control over the entire delivery architecture the content broker is able to allow the content provider to maintain full control of its content. This added control also allows CDNs to be used to deliver dynamic content and streaming media [62]

In practice, CDNs use both reactive and proactive replication. In contrast to reactive approaches, used by proxy caches, proactive replication is extremely beneficial for networks that may suffer from substantial network delays, low bandwidth, or even unidirectional links. For example, in a network with a unidirectional satellite link and no uplink, data may be pushed to a local server, from which local clients can retrieve the data; or a movie that is about to be released can be proactively replicated to multiple servers (avoiding a single server to become overloaded at the release). This approach can be further improved by disseminating content to servers at times when the network is less utilized.

Akamai[1] is the largest and best known CDN. In August 2006 it deploys 20,000 servers, spread over 1,000 networks, located in 71 different countries. However, there are other commercial CDNs deployed that use many fewer servers. Also, some larger corporations choose to set up private CDNs over which they provide training, distribute tools, information and software, as well as provide an infrastructure for efficient wide-area meetings, while saving considerable network resources [166]. In an attempt to scale beyond the limitations set by individual content brokers, without impacting the privacy of each CDN, some research efforts have investigated interconnection of independent CDNs [58, 80].

A common goal for all CDNs is to provide an architecture that improve the overall client experience. When redirecting requests it is therefore important that content brokers provide an infrastructure and mechanism that is transparent for the end user. In particular, client requests should be transparently redirected to an appropriate server. Optimally, the clients should benefit from being redirected while interacting with the system in exactly the same way as if there were only a single server. Many redirection techniques have been proposed for CDNs [21]; however, most commercial systems use some form of Domain Name System (DNS) redirection [103]. For example, Akamai implements its own DNS service using a two level server hierarchy [7].

[1] Akamai, http://www.akamai.com/, August 2006.

1.1.2.3 Peer-to-Peer and Peer-assisted Systems

In peer-to-peer and peer-assisted systems peers distributed across the network contribute to the collective resources of the system. Even though the original Internet was designed on peer-to-peer principles, it is not until the last few years that peer-assisted systems have been considered for content distribution. As more peers choose to share their content and resources, the capacity of these architectures grows. With appropriate techniques to discover nearby replicas, these systems also have the potential to reduce network bandwidth usage.

The main application of current peer-to-peer systems is file sharing among peers. This application is not only the most widespread application, but also the most controversial application. Systems such as Napster [124], Gnutella[2], Freenet [47, 188], Kazaa[3], and many of the sites providing support for the BitTorrent [48] download protocol have gained a multitude of attention from authorities, copyright protectors, and media due to the enormous amount of copyrighted music and movies that are shared among users across these systems. Other applications include distributed computation, computer gaming, and other collaboration applications.

Measurement studies have observed that peer-to-peer traffic is responsible for a large portion of the bytes transferred across the Internet (e.g., [157]). With increasing peer-to-peer traffic locality aware mechanisms, which allow content to be retrieved from nearby rather than far-away peers becomes more important [145, 98].

Peer-assisted content distribution systems and algorithms have been proposed for both live streaming [39, 45, 95, 102] and for on-demand streaming of stored media files [24, 53, 161]. To achieve streaming, these protocols typically establish relatively long-duration streams from the content source and between peers, as organized into some form of overlay topology. In contrast, with BitTorrent [48] and similar download protocols (e.g., [78, 162]) a client may download a file from a large and changing set of peers, using connections of heterogeneous and time-varying bandwidths. This flexibility is achieved by breaking the file into many small pieces, each of which may be

[2] Gnutella, *http://www.gnutella.com/*, August 2006.
[3] Kazaa, *http://www.kazaa.com*, August 2006.

downloaded from different peers. This approach has also been found beneficial in the context of live streaming [86, 191, 190, 112].

Other work has proposed mechanisms to replicate content [116, 51, 55, 152], search for content (or information) [116, 184, 49, 50, 117, 105, 17, 172], as well as route data (or queries) [38, 39, 193, 168] in various types of overlay peer-to-peer structures.

1.2 Problem Description

This thesis considers the scalability and performance of download protocols, used to effectively disseminate data to a large number of requesting clients. In particular, new protocols and policies are designed and evaluated for three different contexts, each achieving scalability through service aggregation and/or replication.

First, this thesis considers the problem of devising single server protocols that minimize the average or maximum client delay for downloading a single file, as a function of the average server bandwidth used for delivery of that file. An equivalent problem is to minimize the average server bandwidth required to achieve a given average or maximum client delay. This equivalent perspective is sometimes adopted. Although delivery of multiple files is not explicitly considered, note that use of a download protocol that minimizes the average server bandwidth for delivery of each file will minimize the average total required server bandwidth for delivering all files as well.

Secondly, this thesis considers the problem of devising policies to select which replica should serve each request, in systems exploiting both service aggregation and replication. Such policies must take into consideration the basic tradeoff between locality of service (maximized by selecting the nearest replica), and efficiency of use of server resources (maximized by selecting the replica at which service can be shared among the largest number of clients).

Finally, a peer-assisted environment is considered in which the content is replicated but peers do not utilize service aggregation techniques. For this context, scalable download protocols, such as BitTorrent [48] have already been proposed, successfully deployed, and have shown to provide good performance [111, 134]. This thesis considers the problem of using adaptations of these download protocols to provide on-demand streaming of stored media.

1.3 Contributions

The main contributions of this thesis are as follows.

- New scalable download protocols are designed for download of a large file from a single server, using a multicast based approach, and their performance evaluated against new analytic bounds on the best achievable performance.
- The relative performance of classes of replica selection policies, of varying complexities, are compared in a context where a large file may be downloaded from multiple replica sites, each using multicast.
- A peer-assisted protocol is designed that splits a large media file into small pieces, uses a piece selection policy to determine which piece to be downloaded next from multiple content server(s) and/or other clients having retrieved part of the file, in an order that allows streaming, as well as a rule to determine when playback can safely begin.

For each of the above contexts a number of abstractions are developed, within which protocols and policies are evaluated. The following sections elaborate on the contributions made in each context.

1.3.1 Scalable Download from a Single Server

To evaluate the performance of existing and new protocols lower bounds on the average and maximum client delay for completely downloading a file, as functions of the average server bandwidth used to serve requests for that file, are developed for systems with homogeneous clients. The results show that neither optimized versions of cyclic multicast nor batching consistently yield performance close to optimal. New, relatively simple, scalable download protocols are proposed that achieve within 15% of the optimal maximum delay and 20% of the optimal average delay in homogeneous systems. Similar to cyclic multicast, these protocols allow clients to start listening to on-going multicasts at the time of their arrival, but limit server transmissions to time periods in which (probabilistically) there are more clients listening.

For heterogeneous systems in which clients have widely-varying achievable reception rates, an additional design question concerns the use of high-rate transmissions, which can decrease delay for clients that can receive at such rates, in addition to use of low-rate transmissions that can be received by all clients. A new scalable download protocol for such systems is proposed, and its performance is compared to that of alternative protocols as well as to new lower bounds on maximum client delay. The new protocol achieves within 25% of the optimal maximum client delay in all scenarios considered.

Throughout this analysis it is assumed that each requesting client receives the entire file (i.e., clients never abort their request while waiting for service to begin or after having received only a portion of the file). The analysis and protocols presented are compatible with erasure-coded data. Each client is assumed to have successfully received the file once it has listened to multicasts of an amount of data L (termed the "file size", although with packet loss and erasure coding, L may exceed the true file size). Poisson request arrivals are typically assumed, although generalizations are discussed in some cases. Note that Poisson arrivals can be expected for independent requests from large numbers of clients (during time periods with constant arrival rates). Furthermore, multicast delivery protocols that have high performance for Poisson arrivals, have even better performance under the more bursty arrival processes that are typically found in contexts where client requests are not independent [68].

1.3.2 Scalable Download from Multiple Servers

In large distributed systems implementing both replication and service aggregation, a basic tradeoff is between locality of service (maximized by selecting the nearest replica), and efficiency of use of server resources (maximized by selecting the replica at which service can be shared among the largest number of clients). Rather than propose a specific policy to mediate this tradeoff, classes of policies of differing complexities are compared within the context of a simple cost model, capturing both the service requirements of the individual replica servers, and the additional cost associated with retrieving service at remote replicas.

A large popular file is assumed to be replicated at multiple servers across the network, from which the file can be downloaded. The set of servers with a replica may be determined based on expectations of future demands, availability, or some other system requirements. Here, the set of servers with a replica of the file is assumed to be predetermined. It is further assumed that each server implements some form of service aggregation technique allowing multiple client requests to be served together, rather than individually.

Within each class of policies, limits on the best achievable performance are determined (or representatives defined) for both batching and cyclic multicast aggregation approaches. When using cyclic multicast the file is assumed to be erasure encoded. Similar to the analysis used for the single server case, this analysis assumes that requests arrive according to a Poisson process, and no client aborts their request while waiting for service to begin or after having received only a portion of the file.

It is concluded that (i) selection using current system state information (rather than only proximities and average loads) can yield large improvements in performance, (ii) when it is possible to defer selection decisions (e.g., when requests are delayed and served in batches), deferring decisions as late as possible can yield additional large improvements, and (iii) relatively simple policies using only "local" (rather than global) request information are able to achieve most of the potential performance gains.

1.3.3 On-demand Streaming using Scalable Download

Based on the design of the relatively simple and flexible BitTorrent download protocol, this thesis proposes a peer-assisted BitTorrent-like approach to media file delivery which is able to achieve a form of "streaming" delivery, in the sense that playback can begin well before the entire media file is received. Achieving this goal requires: (i) a piece selection strategy that effectively mediates the conflict between the goals of high piece diversity (achieved in BitTorrent using a rarest-first policy), and the in-order requirements of media file playback, and (ii) an on-line rule for deciding when playback can safely commence.

Candidate protocols including both of these components are presented and evaluated using event-based simulations, in which each peer is assumed to be bottlenecked by either its upload or download rate. Locality is not considered in this part of the thesis. It is further (very conservatively) assumed that no peer, except the origin content source, shares pieces once it has received the whole file. In a real system, peers are likely to continue serving other peers as long as they are still playing out the media file, while other peers may (graciously) choose to upload to other peers beyond that time. With the higher availability of rare pieces and download bandwidth in such systems, the benefits of more aggressive piece selection techniques (giving priority to earlier pieces rather than rare pieces) are likely to be even greater than presented here.

It is found that simple probabilistic piece selection policies, giving preference to earlier pieces, allow peers to begin playback well before the entire file is downloaded. Further, whereas no on-line strategy for selecting startup delays is expected to give close to optimal startup delays (without significant chance of playback interruption), promising results are obtained using a simple startup rule. Before starting playback, the rule requires the retrieved number of pieces to exceed some (small) threshold, and the rate at which in-order pieces are being accumulated to exceed a value sufficient to allow continuous playback without interruption, if that rate was to be maintained.

1.4 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 reviews related work, outlining the current state of scalable download protocols and setting existing solutions into context. Chapter 3 develops lower bounds and new scalable download protocols that achieve close to optimal performance when downloading large files from a single server. Chapter 4 considers the problem of replica selection in systems exploiting both replication and service aggregation. Chapter 5 proposes adaptations of existing scalable peer-assisted download protocols, in a way that allows on-demand streaming. Conclusions and directions for future work are presented in Chapter 6.

Chapter 2
Background

Rather than attempting to provide a complete survey of all existing scalable content delivery protocols and architectures, this chapter focuses on the techniques most relevant for the three contexts considered in this thesis. Section 2.1 presents an overview of various approaches to implement multicast. Section 2.2 surveys previous work on scalable single server download protocols that use multicast-based service aggregation. Section 2.3 discusses replica placement and selection techniques applicable for the context of scalable download from multiple servers. Finally, related work on peer-assisted content delivery protocols is surveyed in Section 2.4.

2.1 Multicast

When distributing content to multiple clients across the Internet, content servers have traditionally used multiple concurrent unicast connections. This approach suffers from highly redundant usage of network resources and high server overhead. With multicast, in contrast, a single transmission of the content can be received by multiple receivers. A collection of nodes interested in receiving the same multicast transmission is called a multicast group, and the server and network resources used to transmit each multicast to each member of the group is called a multicast channel. Throughout this thesis, “listening to a channel” refers to listening to a particular on-going or intermittent multicast transmission. Content is disseminated using a multicast delivery tree, and in contrast to replication strategies, multicast does not require any persistent storage capacity in the network. Multicast significantly decreases the bandwidth requirements at the server, and decreases the total bandwidth required throughout the network.

Despite first being implemented as an overlay system implemented at the application-level [70], multicast was originally envisioned as a network-level functionality (“IP multicast”) supported by the network routers [59]. In theory this would give shorter paths and use less total network bandwidth; however, for numerous reasons IP multicast has seen slow commercial deployment [64, 87]. This has prompted much research on implementing multicast at the application level. This section will discuss both IP multicast and application-level multicast.

2.1.1 IP Multicast

This section discusses how the current multicast solution has evolved and concludes with a discussion of current deployment issues and alternative network-level solutions that have been proposed.

In the traditional IP multicast service model, a multicast group is formed by a set of clients that have all expressed interest in receiving transmissions sent to some particular multicast address (used as the group identifier). While only the nodes currently in the group receive data sent to the multicast address, any node, including nodes that are not members of the group, can send data to the group by addressing transmissions to the multicast address. Implemented at the network-level, the content is delivered without guarantees of in-order or loss-free delivery.

The Internet Group Membership Protocol (IGMP) [32] provides the functionality to handle group membership. It operates between clients and their directly attached routers. Group members use this protocol to inform their nearest router about multicast groups which they wish to join or leave. Note that IGMP is only used for group membership, and other protocols called multicast routing protocols are needed to build and maintain delivery trees for each group.

To achieve scale and administrative autonomy the Internet is organized into domains or regions, each called an Autonomous System (AS). Routing protocols are generally categorized as either *intradomain* routing protocols, responsible for routing within a domain, or *interdomain* routing protocols, responsible for routing between different domains.

Many different intradomain protocols have been proposed for multicast routing within an AS. The main differences among these protocols concern how they build and maintain the multicast tree structure. These routing protocols are normally categorized as either *dense-mode* protocols or *sparse-mode* protocols. Dense-mode protocols are designed to perform best when multicast transmissions must pass through most of the network routers and normally use some form of "broadcast and prune" mechanism. Sparse-mode protocols are designed to perform best when multicast transmissions need to pass through only a small fraction of the network routers and rely on receivers explicitly sending requests to join the multicast group. Dense-mode intradomain multicast protocols include Distance Vector Multicast Routing Protocol (DVMRP) [178] and Multicast Open Shortest Path First (MOSPF) [123], while sparse-mode protocols include Core-Based Trees (CBT) [19, 18] and Protocol Independent Multicast Sparse Mode (PIM-SM) [60, 61]. PIM-SM is also an integral component of the current interdomain multicast architecture.[1] PIM-SM forms a reverse shortest path tree, rooted at a rendezvous point (RP) associated with a multicast group, by setting up routing states at routers when propagating the explicit join messages towards the RP. The tree is a reverse shortest path tree in the sense that the path from each receiver to the RP uses the "shortest" IP path; however, with the asymmetry of path lengths, these paths are not necessarily the shortest paths from the RP to each receiver. A novel feature of PIM-SM is its ability to let receivers switch from group-shared trees (in which all content is forwarded through the RP) to source-specific trees (in which the multicast tree is rooted at the content source). This ability can improve performance for clients and offload the RP.

While all routers in a specific AS generally deploy the same multicast routing protocol, routers in different domains may use different protocols. Therefore, interdomain routing protocols are generally required to achieve interoperability among domains using different routing protocols. Up until the beginning of 1999, DVMRP was almost exclusively the only protocol deployed for interdomain routing. However, as a dense-mode protocol (using a broadcast and prune approach) it is not suited for sparse sets of participating routers, and as observed by Rajvaidya and Almeroth [137], DVMRP was almost entirely replaced in March 2000. In the replacement multicast architecture, PIM-SM is used for routing and the Multiprotocol Boarder Gateway Protocol (MBGP) [22], which extends the Boarder Gateway Protocol (BGP) [144], is

[1] PIM-SM is currently the only multicast routing protocol used for interdomain routing and it (or other PIM versions) is also typically used for intradomain routing [160].

used to exchange routing information among multicast-enabled domains. MBGP allows domains to learn of paths to reachable multicast-enabled networks.

The Multicast Source Discovery Protocol (MSDP) is currently used in conjunction with MBGP and PIM-SM, although it is viewed as a short term solution. MSDP uses flooding among RPs of different domains to distribute information about new sources that have started transmitting to a multicast address. If a RP has received a join message from within its domain for a multicast group to which a new external source is transmitting, the RP sends a join message to the source. Any data received by the RP will then be forwarded on the local multicast tree. With knowledge of the source, source-specific trees can be established using PIM-SM. Almeroth discusses the MBGP/PIM-SM/MSDP solution and some of its drawbacks [11], while other work has proposed interdomain solutions to overcome MSDPs scaling problems [106].

While quantifying the bandwidth savings at the server from use of multicast is relatively easy, quantifying the bandwidth savings throughout the network is more difficult. For example, each network link may have a different cost function associated with it and costs may be associated with many different organizations. Chuang and Sirbu [46] define a cost measure, originally aimed at pricing multicast, as the ratio of the total number of multicast links in a distribution tree and the average path length between two arbitrary nodes in the network. Through extensive simulations of shortest path multicast trees using both real and generated network topologies they found this ratio follows a power law, scaling as a power of the number of receivers. The same power law was also found using analysis of k-ary trees [128]. Van Mieghem *et al.* [175] perform a more thorough analysis, finding that the exponent in the power law increases with the number of nodes in the system. Chalmers and Almeroth [41] validate this later model using data obtained from measurements of real multicast trees, and explain it by the underlying network connectivity putting a constraint on the possible shapes of interdomain multicast trees. While previous studies [46, 128] suggest a power law exponent of about 0.8, Chalmers and Almeroth [41] indicate that it may be between 0.7 and 0.8 for real networks. They also observe a bandwidth reduction of 55-70% from using multicast, with multicast groups as small as 20-40 receivers.

Despite considerable potential bandwidth reduction, IP multicast has not been widely deployed outside individual organization-based networks [155]. While there are many difficulties with implementing wide-area multicast at the network level [64, 87, 11], most of which are due to the highly heterogeneous nature of the Internet (with many domains deploying different routers, protocols, etc.), one of the main reasons often used to explain this fact is the inherent complexity of implementing the IP multicast service model, in which both multiple senders and receivers are allowed. By restricting the model to a single sender (or source), as in the Single Source Multicast, deployment may be made much easier [87, 63]. Here, the root of the tree is placed at the sender and each receiver specifies both an IP multicast address (specifying the group) and a regular IP-address (specifying the source). Other researchers have proposed leveraging from existing unicast solutions [170, 52, 138]. For example, Ratnasamy *et al*. [138] propose an approach in which routers use BGP routing tables to compute dissemination trees, consisting of the union of all unicast paths to know receivers, and forward data only to the next set of routers in the dissemination tree [138].

IP multicast has also been deployed using short-term solutions, which satisfies short-term demand, but makes re-deployment difficult. For example, IGMP did not support Single Source Multicast until late 2002 [32], and this new IGMP version is still constrained by backwards compatibility with previous versions. With observations of successful bandwidth savings [155], improved stability of the current multicast infrastructure [137], and increasing demands for the original service model (e.g., distributed network games), the feasibility of wide-area multi-source deployment of IP multicast is currently being revisited [138].

2.1.2 Application-level Multicast

The end-to-end principle [154] states that complexity should be pushed to the end systems, keeping the core of the network simple. This is done in application-level multicast. In this approach, network-level routers play no role in implementing multicast. Instead, multicast is implemented by application-level software that establishes conventional unicast connections among a collection of nodes, including those that wish to receive the multicast transmissions, the source(s), and possibly other nodes as well. Data transmitted by a source is relayed by these participating nodes, across unicast connections, until reaching all receivers. The collection of nodes and the connections (“links”) between them is commonly called a “virtual”, or “overlay” network. Some of the main advantages of application-level multicast are: (i) billing can be done on participants in the virtual network, (ii) group management can be handled at the application-level, (iii) higher level services can benefit from well understood unicast solutions, (iv) multicast address allocation can be done in a single virtual domain, (v) existing unicast tools can be used to monitor the overlay network, and (vi) interdomain routing is avoided by routing in a single virtual domain. Since application-level multicast can be implemented by any willing collection of nodes, without requiring additional infrastructure, it is also relatively easy to deploy.

Depending on the construction of the routing overlay, for the control and data distribution, application-level multicast protocols can be categorized into three classes: (i) *mesh-first* approaches, (ii) *tree-first* approaches, and (iii) *key-based* routing approaches in which the multicast tree is created on top of structured overlays.

Mesh-first protocols, such as Gossamer [42] and Narada [45], perform a two-step process in which a mesh of connections is created before the actual delivery tress. Both Gossamer and Narada use delay measurements and threshold algorithms to create and maintain a sparse mesh with limited out-degree at each participating node. The redundancy of the mesh provides for better reliability than a tree (where the breakage of a single link partitions the network) and mesh maintenance can ensure that the mesh links in use have relatively low end to end delays and high-bandwidth. To form their respective delivery trees using the mesh network links, both protocols use a distance vector protocol, in which nodes periodically exchange with their neighbors their established network distance over the mesh to each destination, thus allowing each node to determine their best next hop (and correct estimates of network distances) to each destination. By building their delivery trees on top of a mesh they inherit the attractive properties of the mesh, while permitting use of a relatively simple tree building protocol. While Narada is tailored for smaller end-system groups [44], Gossamer achieves scalability by adding a two-level hierarchy to its design.

For the purpose of robustness tree-first protocols may also create a mesh; however, central to this approach is that protocols begin by creating a group-shared or source-specific tree. ALMI [127] is a tree-first protocol tailored towards multicast groups of small sizes using group shared trees, Yoid [74] and Host Multicast Tree Protocol (HMTP) [189] are more scalable protocols using group-shared trees, while Overcast [95] is a scalable single source protocol using source-specific trees. In ALMI, a central session controller uses the relative distances of the different nodes to form a minimum spanning delivery tree. With the more scalable protocols, each node is responsible for finding an appropriate parent on the tree. A potential parent should have sufficient resources to support an additional child, and the addition of a link to the potential parent should not result in a routing loop. Other implicit protocols structure themselves into a hierarchy which implicitly defines the delivery tree. For example, NICE [20] structures itself into a hierarchical overlay of clusters, where each cluster has a cluster leader that is also a member of a higher level cluster. This structure allows NICE to efficiently route messages to all nodes in the overlay by having each node send the messages to all nodes in all the clusters in which it participates.

Key-based approaches build multicast trees on top of structured overlays that use key-based routing [56, 192, 151, 37, 169, 141, 131], wherein all nodes are given identifiers in a numerical key space, and routing proceeds using some technique that guarantees movement closer in the key space to the destination node at each hop. Examples of such protocols are Bayeux [193], Scribe [153, 38], SplitStream [39], and CAN-multicast [139]. These protocols all take advantage of the underlying key-based routing (KBR) [56] mechanism, when forming their distribution trees.

Much of the work on application-level multicast considers a scenario in which a different overlay network is formed for each multicast group, consisting only of the sender and receivers (e.g., client work stations) for that group. It has also been suggested that application-level multicast could be offered as an infrastructure service, using an overlay network constructed using servers distributed through the Internet in a manner of a CDN [42, 43].

Despite numerous advantages, application-level multicast uses more network bandwidth than IP multicast. This is primarily since different links in an overlay network may share links in the underlying physical network, hence requiring multiple copies of the same content to traverse the same network links. Chu *et al*. [45] define a metric they call Normalized Resource Usage (NRU), as the ratio of the network usage for a content delivery scheme of interest relative to that for IP multicast. Assuming symmetric links and in light of the previously discussed findings of Chuang and Sirbu's [46], the expected NRU for sequential unicast is $O(N^{0.2})$, where N is the number of receivers in the distribution network. Unfortunately, not many researchers use this metric when evaluating their protocols. For application-level multicast to save network resources, compared to sequential unicast, it should have an NRU better than $N^{0.2}$. Note, however, that even when application-level multicast does not reduce substantially the network bandwidth usage, compared to unicast, it still offers substantial bandwidth savings at the server.

Other measures that have been used to evaluate application-level multicast protocols are the stress and stretch metrics. These metrics measure the number of identical packets sent over the same physical link, and the ratio of the path length along the delivery tree and the length for the unicast path, respectively. Note that the stress for IP multicast is always 1 but is generally higher for application-level multicast. Assuming symmetric routes in the physical network (i.e., the route from A to B is the reverse of the path from B to A), IP multicast will deliver data along the unicast paths, resulting in a stretch of 1. This property is not necessarily true if the routes are asymmetric since the direct unicast path may not be the same as the reverse shortest path, used to create many multicast trees (e.g., PIM-SM). Further, research suggests that a substantial portion of IP-routes on the Internet are non-optimal and efficient routing choices over a virtual overlay can achieve a stretch below 1 [158, 159, 13, 14]. Because of asymmetries and non-optimal IP-routes, application-level multicast certainly has room to achieve a lower stretch than IP multicast.

2.2 Multicast-based Single Server Scalable Download Protocols

This section discusses the main multicast-based download protocols used for content delivery from a single server. It is important to note that the multicast-based protocols considered here are beneficial even for systems in which some form of multicast-like operation has been implemented only at the server, and not in the network itself. In this case, server resource use (such as disk access bandwidth, CPU and memory use) can be reduced by replicating the data at (or just before reaching) the network interface [150]. Although this may result in the network interface becoming a bottleneck it can significantly improve the performance of download systems in which some other server resource is the bottleneck.

Existing multicast-based approaches for scalable download of popular files from a single server include *batching* [66, 182] and *cyclic multicast* [10, 150]. With batching, clients wait to begin receiving a requested file until the beginning of its next multicast (or broadcast) transmission, which collectively serves all of the waiting clients that have accumulated up to that point. With cyclic multicast, the file data is cyclically transmitted on a multicast channel that clients begin listening to at times of their choosing, and continue listening to until all of the file data has been received. Section 2.2.1 and 2.2.2 discuss batching protocols and cyclic multicast protocols, respectively. Hybrid protocols are discussed in Section 2.2.3. The accommodation of heterogeneity in multicast-based protocols is considered in Section 2.2.4.

2.2.1 Batching Protocols

Considerable prior work has concerned scheduling one or more broadcast channels that serve a collection of small, fixed length objects using a batching approach [66, 182]. The main problem considered is that of determining which object should be transmitted on the channel (or channels) at each point in time, so as to minimize the average client delay. With batching protocols scheduling small files, the average client delay is often defined as the time that a client waits for service, from its arrival until it first starts receiving service. This delay is often referred to as the access delay. Both push-based [12, 83, 2] and pull-based [66, 182, 8] protocols have been proposed. Push-based protocols determine a transmission schedule based only on average object access frequencies. Pull-based protocols assume knowledge of the currently outstanding client requests. Hybrid approaches that combine push and pull are also possible [3, 167]. Other work has investigated batching protocols for streaming rather than download [6, 57, 173].

2.2.1.1 Push-based Protocols

In asymmetric environments, where clients have little or no uplink bandwidth, directly communicating requests may not be feasible. For such environments push-based protocols are commonly used. Push-based protocols determine a transmission schedule based only on average object access frequencies, obtained using some offline method, model, or other estimation of the expected demands. Given knowledge of only the average access frequencies, a periodic delivery schedule is optimal [12]. Providing a natural relaxation of the problem, where scheduling conflicts are ignored, the optimal spacing between transmissions, as well as a lower bound, can be derived. Hameed and Vaidya [83, 84, 174] generalize these results to consider different file sizes and transmission errors. They further suggest a simple but close to optimal scheduling algorithm, inspired by packet fair queuing. The scheduling time of this algorithm scales logarithmically with the number of files and is extended to handle multiple channels.

Various other approximation algorithms have been proposed that provide performance guarantees. For example, Kenyon *et al*. [100] propose a polynomial-time approximation scheme (PTAS), which separates files into three categories and places them in a fairly intuitive manner. Transmissions of the most frequently requested files are first scheduled in a near optimal fashion over the schedule space provided for the two first categories. Secondly, transmissions of the files belonging to the largest group of files are scheduled in a round robin fashion, before the schedule is finally stretched to give room for the leftover files with the lower cost.

Another approach proposed in this context is the *broadcast disks* technique [2], in which files are partitioned into groups with similar access frequencies. Placing files with similar access frequencies on a single disk minimizes the required storage, and allows each individual disk to operate using a round robin schedule. A transmission schedule for the channel is created by separating each disk into smaller chunks and multiplexing among transmitting chunks from each disk, based on their relative access frequencies. Since the times of future transmissions are fixed, this scheme also has the added benefit of simplifying client operation.

Push-based protocols do not utilize potential information about outstanding requests, and are therefore not suitable for content delivery in on-demand systems with explicit knowledge of outstanding requests.

2.2.1.2 Pull-based Protocols

Pull-based protocols employ a queue of client requests, and a policy for determining which request(s) in the queue to serve next by transmitting the requested object. Early papers by Dykeman *et al*. [66] and Wong [182] consider policies such as: First Come First Serve (FCFS) – broadcasts the file that has the request with the longest individual wait time; Most Request First (MRF) – broadcasts the file with the maximum number of pending requests; Most Request First Lowest (MRFL) – same as MRF but breaks ties in favor of the file with the lowest access probability; and Longest Wait First (LWF) – broadcasts the file with the largest cumulative waiting time over all pending requests for that file. Among these policies, LWF results in the lowest average access times (when averaged over batches), MRF in the lowest access times for popular files, and FCFS in the most fair access times. The criteria used by FCFS and MRF are combined in the RxW policy, proposed by Aksoy and Franklin [8]. When deciding which object to transmit next, this policy weighs the number of pending requests (R) for each file and the associated longest waiting time of the pending requests (W). RxW has been shown to provide a relatively good tradeoff between the fairness of FCFS and the bias towards popular requests in MRF.

Acharya and Muthukrishnan [4] consider the case of varying file sizes and allow for preemption. Focusing on the completion time of requests, rather than the access time, they argue that the user-perceived performance decreases less per time unit for clients downloading large files than for clients downloading smaller files, as their expected download time is larger and the perceptual difference becomes smaller. Based on this observation they define the stretch factor as the ratio of the completion time of a request and its required service time (if the request was served immediately without any preemption). By minimizing the total stretch, rather than the completion times, it is suggested that fairness among all jobs can be maintained, while minimizing some "perceived" client delay. It is proposed that each file is broken into smaller segments so that transmissions of larger files can be interrupted, allowing smaller files to keep their stretch factor low, and improving both the overall stretch and the average completion times.

Batching protocols do not typically allow clients to join an on-going multicast, instead clients are required to wait until the beginning of the next multicast transmission; clearly, when considering delivery of large erasure encoded files this is sub-optimal. Further, previous work on batching protocols assumes that the system devotes a fixed set of server resources to file delivery, and does not consider the case where the server bandwidth devoted to this task may be somewhat elastic (and the system may adjust its resources usage based on current demands).

2.2.2 Cyclic Multicast Protocols

Prior work on scalable download of large files from a single server has focused on cyclic multicast, in which a file's data is cyclically transmitted on a multicast/broadcast channel [146, 10, 26, 176, 31, 150, 27]. Each requesting client can begin listening to the channel at an arbitrary point in time, and continues listening until all of the file data has been received.

Now, consider a simple cyclic multicast protocol using only one channel and one object. As the data is cyclically transmitted on the channel a client can tune in and listen until it receives the whole object. Assuming there are no losses, this scheme is very efficient since each client is served immediately and only has to wait for the duration of one broadcast, no matter when the client arrives. Unfortunately, this is an unreasonable assumption on some networks. Packet losses do occur and clients missing a packet would either have to listen to the channel until the missing packet gets retransmitted in a later cycle or rely on some additional mechanisms to retrieve the missing data.

Erasure codes have been proposed to accommodate packet losses. As in Section 1.1.1, let N be the number of blocks in the original file and M the number of encoded blocks. With the simplest form of erasure codes, one additional block ($M = N+1$) is created using the exclusive-or (XOR) operation on the N original blocks. This operation is performed on a bit-by-bit basis. Having received any N out of these $N+1$ blocks ensures that the $(N+1)^{th}$ block can be retrieved as well, thus allowing the original file to be reconstructed. This approach can be extended in a number of ways; however, the stretch factor M/N of these simple schemes is typically small, and for many extensions receiving N distinct blocks does not ensure that the original blocks can be reconstructed.

By multiplying the original source blocks with a transformation matrix of size $M \times N$, consisting of linearly independent rows, Reed-Solomon codes can be created for any arbitrary stretch factor M/N [148]. Given a set of N distinct blocks, the original source blocks can be obtained by multiplying the inverse of a transformation matrix (created by a subset of the rows in the original transformation matrix) with the received blocks. Unfortunately, matrix inversion is costly and this approach suffers from a serious scalability problem, as the objects become bigger and N grows large.

Tornado codes [114], LT codes [115], and Raptor codes [164] allow for faster decoding. These coding techniques require the receiver to receive on average $(1+\varepsilon)N$ distinct blocks before decoding is possible, where ε is a small number. For example, with Tornado codes where ε is typically in the range 0.05 to 0.1. Similar to some of the basic schemes discussed earlier, both encoding and decoding is efficiently performed using only XOR operations. However, here each encoded block is created using a linear combination of original source blocks. Similarly, given a set of encoded blocks, the original source blocks can be decoded (or reconstructed) by considering each block as an equation and solving these equations in an order based on the number of unknown source blocks in each equation.

In comparison to Reed-Solomon codes, Tornado codes have far superior decoding speeds. However, they require a large amount of coding and decoding information to be communicated to both servers and clients. In addition, the encoding and decoding memory requirements at both the server and the clients are proportional to the object size multiplied by the stretch factor [115]. By using random number generators, rather than locally stored transformation information, LT codes can achieve a dynamically expandable stretch factor, and significantly decrease the memory and storage requirements on the clients and servers. Raptor codes [164] are an extension of LT-codes, which achieve linear time encoding and decoding.

Using the erasure codes described above, Byers *et al*. [31] envision a content distribution system (termed a digital fountain) in which the content provider provides clients with an unbounded stream of distinct encoded data blocks to which clients can tune in until it has received enough data to reconstruct the file. In the ideal case each block is of full value to the receiver. Having received some arbitrary set of data with the combined size equal to the size of the original data the client should, with minimal effort, recover the original data; i.e., obtaining the same satisfaction as if the original content was delivered directly to the client. Based on this ideal case, the perfect erasure code should (i) have an infinite stretch factor, (ii) have an infinitesimal encoding and decoding cost, and (iii) allow any receiver to decode the data after receiving exactly N unique blocks. As discussed above there are no known erasure codes with these properties. However, due to their superior decoding efficiencies Raptor codes are considered to be the codes that are most efficient with today's technology. In fact, these codes are used in Digital Fountain's commercial systems.[2]

2.2.3 Hybrid Protocols

There has been some prior work on hybrid protocols that combine batching and cyclic multicast, specifically the work by Wolf *et al.* [180]. The focus in the work by Wolf *et al.* is on delivery of digital products using spare bandwidth in a broadcast television system, and thus their algorithms assume a fixed schedule of broadcast channel availability and fixed delivery deadlines with associated delivery payments.

Client requests for a particular file are allowed to be aggregated using techniques similar to those used in cyclic multicast protocol. However, as with size-based approaches [4], files are split into subtasks. The subtasks for a particular file are scheduled in cyclic order, allowing requests waiting for the same file to be served as a single batch. The next subtask to be scheduled is reevaluated at the completion of a subtask. Rather than using FCFS, or some other common techniques, to decide which subtask to schedule next, the authors design scheduling techniques that attempt to maximize revenue.

They assume that each request is associated with delivery deadlines and corresponding delivery payments. This defines a revenue function indicating the profit as a function of the completion time. Transmissions (of cyclically enumerated subtasks for each requested file) are scheduled in such a way that the total revenue of an on-

[2] Digital Fountain Inc., *http://www.digitalfountain.com/*, August 2006.

demand delivery system is maximized. After demonstrating that the scheduling problem is NP-hard they propose three heuristics to determine which subtask to schedule next. The most complex of these heuristics weighs the remaining delivery time of the file, the time until the different delivery deadlines, and the cost of missing each deadline. The second heuristic relaxes this approach by only considering the cost of missing the next deadline that is feasible for each object. The third heuristic greedily maximizes the profit per time unit of each possible scheduling. Of these heuristics, the first performs the best and the greedy approach the worst. However, all three heuristics are shown to outperform a non-hybrid protocol based on a transportation problem formulation, which does not split the file into subtasks.

Similar to this work, Chapter 3 of this thesis designs hybrid protocols that combine elements from both cyclic multicast and batching protocols to achieve superior performance. However, in contrast to this work, this thesis assumes complete flexibility in when transmissions occur, and develops protocols that achieve near-optimal average or maximum client delay as a function of the average required server bandwidth.

2.2.4 Client Heterogeneity

Real deployable delivery systems generally serve clients that are spread over many domains with highly diverse characteristics. For example, the available bandwidth, round trip times and loss rates may be very different between different clients. Various multicast-based protocols have been proposed for a heterogeneous client population, using one of two main approaches. The first, and simplest, approach is to categorize the client population into a set of groups. For each group a different version of the content is encoded and delivered over the network. Clients simply choose to receive the version that best fits their respective channel characteristics. Although it is possible to allow clients to switch between versions during delivery, this may create delivery interruptions and/or redundant data being received.

A second approach is to deliver a single version of the file, but with transmissions spread over multiple channels. Each client listens to the subset of channels appropriate to its achievable reception rate [122, 176, 29, 107, 113]. By careful selection of the order in which data blocks are transmitted on each channel [26, 27], or use of erasure codes with long stretch factors (i.e., $M >> N$) [164], receptions of the same data block on different channels can be reduced or eliminated. A client can adjust to current network conditions by changing the set of channels it is listening to.

2.3 Server Placement and Selection

By strategically placing replica servers, the scalability and efficiency of CDNs and other content delivery systems using replication can be improved, especially if combined with an efficient replica selection strategy. This section overviews prior work concerning where to place replica servers, what content to store at each replica server, as well as how to select the appropriate replica server to serve each client request. Although these questions can be considered separately the design choice of one component will generally affect the other choices. Note that these choices also are affected by many other factors, such as the specific client, content and network characteristics. Throughout this thesis, a "replica" refers to a replica server that has a copy of the replicated file. Note that a replica server can be a replica for many different files.

2.3.1 Replica Placement

Depending on the CDN, the relationship between placement of replica servers and content is different. A large content broker, such as Akamai, with tens of thousands of servers distributed over the Internet generally only stores each file (to be replicated) at a smaller subset of its servers [62]. In contrast, a smaller CDN, possibly custom made for a particular client population, may replicate the content to all its servers.

Although two different tasks, both placement of servers and content can generally be abstracted in the same way. Both consider the problem of deciding where to place service resources to satisfy a given demand. In the case of content placement, demand is for the particular content replicated. In the case of server placement, the demand that must be satisfied is the cumulative demand over all content served by the system. However, server placement is more costly as it requires investing in additional infrastructure. Therefore, server placement is generally done incrementally, while content placement is done with more freedom as replicas can easily be distributed to any subset of servers provided by the CDN.

Simple iterative greedy approaches that place one server or file at a time have proven to achieve similar network costs as optimal server or file placement [133, 104, 94, 135]. For both server and content placement, increasing the number of servers (or replicas) only results in significant reduction of server load or download times when the number of servers (or replicas) is relatively small [94]. With increasing number of servers, diminishing returns are observed. However, with increasing load on each server more replicas may be required to offload current replicas, and avoid congestion.

By minimizing the overall network delay suffered by the client population Cameron *et al*. [33] investigate the relationship among demand, delay, and optimal server/replica placement. Analytic expressions and simple algorithms are derived to determine the number of replicas to allocate to each file, as well as where these replicas should be placed. Wang *et al*. [179] do not consider where to place the replicas but observe significant improvements in system capacity by dynamically adjusting the number of replicas according to server load and demand.

Reliable application-level multicast protocols, or some other distribution mechanism, can be used to distribute the content to the replica servers (e.g., [43, 95, 108]). Recent research efforts include specialized protocols for reliable replication and content distribution of large files from a central server to multiple replica servers [126, 75]. For example, SPIDER [75] uses multiple dynamic delivery trees and an end-to-end flow control algorithm relying on TCP connections between neighboring overlay nodes.

2.3.2 Replica Selection

Most previous work on replica selection concerns selection among replicas that do not implement service aggregation techniques. In general, such replica selection techniques may attempt to minimize network delays, maximize the download bandwidth, accomplish load balancing among the servers, or achieve some other objective (e.g., Akamai direct requests to closeby replicas that have available resources and are likely to have a copy of the content [62]). However, because some of the above design goals may be conflicting they will have to be weighed against each other. Most current systems give precedence to keeping the network delay low rather than maintaining high bandwidth between the replica and the client.[3] With the introduction of more and more high-bandwidth applications, such as video streaming, this focus is likely to shift.[4] To achieve a good compromise between various objectives a CDN may use some simple heuristics that weigh the importance of various factors when determining to which replica to direct each client's request. Rather than directing clients to some optimal replica, Johnson *et al*. [96] observe that Akamai[5] and Digital Island[6], two of the largest CDNs, primarily attempt to avoid directing clients to bad servers.

Various replica selection techniques have been proposed to select a replica that is "close" to a given client, in terms of delay, hop count, or some other metric. In the simplest solution each client is given a list of all servers and uses probes to determine which replica is the closest. This method is not desirable since it relies on the client to perform tasks it normally would not do, and does not scale to a large number of clients and servers. The IDMaps architecture [93] consists of third party instrumentation boxes, called Tracers, which actively measure distances among themselves to form a distance map. This map can then be used to provide a service indicating which server is the closest. Other approaches cluster clients into groups, based on their distances to different servers [15], use distances to different landmarks [140], or some other set of virtual coordinates (e.g., [54]) to determine which server may be the closest. At this point it should be noted that dynamic replica selection techniques can significantly outperform static replica selection techniques [35]. Anycast is a particularly promising approach, in which a client requesting service sends an anycast message to an anycast address, shared by a set of servers. While the request is directed to all servers with this address, the idea with anycast is that the request is served (or answered) by the "best" replica, specified by some predetermined criteria [186].

The best replica to serve a request may change during the download of a large file. Switching replicas during a download can be costly but it may in many cases be

[3] Note that the round trip time of an end-to-end path and the achieved download rate on that path may be highly correlated. In particular, studies of the TCP protocol have shown that the throughput of long duration TCP flows varies between the inverse and the inverse square of the round trip times [109, 73].
[4] Unfortunately, more advanced tools or techniques are required to measure the available end-to-end bandwidth or capacity (e.g., [92, 65]).
[5] Akamai, *http://www.akamai.com/,* August 2006.

desirable. For example, the advantage of a faster server has to be weighed against the added delay experienced for a client when setting up a new connection [97]. However, a sometimes more efficient technique is to allow clients to retrieve data in parallel from different replicas. Parallel download simplifies replica selection and improves the download rates of individual clients [30, 129, 149, 187]. Clients utilizing parallel download naturally adapt to changing network conditions, and improve their resilience to congestion and network failures. However, as each connection is associated with some overhead, the advantages of parallel download may decrease as the portion of clients using parallel download increases [77, 101, 165].

In addition to the above work, assuming each request is served individually, some work has considered replica selection in systems utilizing both replication and aggregation [9, 72, 81]. All these papers consider the specific context of media streaming and corresponding streaming-based service aggregation techniques. Among these studies, Fei *et al*. [72] consider systems in which a long-lived multimedia stream is being multicast concurrently by replicated servers and the objective is to direct clients to servers so as to minimize the total network bandwidth usage. They show that the replica selection problem in this scenario is NP-complete, and compare a number of heuristic policies. Guo *et al*. [81] design and evaluate replica selection techniques for replicated video-on-demand servers, each with a fixed number of channels. Several heuristic techniques are proposed and shown to outperform a basic policy that always directs requests to the closest replica.

Unlike all of the above replica selection techniques, both the ones using and the ones not using service aggregation, but similar to the work presented in this thesis, Almeida *et al*. [9] assume that each replica server devotes *varying* resources, on-demand, to the service of interest, rather than statically allocating fixed resources. They consider the problem of replica placement and selection/routing with a weighted sum of network and replica server bandwidth usage as the objective function to be minimized, and show that, in the assumed on-demand media streaming context, use of service aggregation can result in optimal solutions that are very different from those for systems without service aggregation.

---

[6] Digital Island was acquired by Cable & Wireless (http://www.cw.com/, August 2006) in June 2001.

2.4 Peer-assisted Content Delivery

A highly scalable approach to content delivery is to utilize the resources of clients. In peer-assisted content distribution systems, clients contribute to the collective power of the system by making (part of) their upload bandwidth (and/or other resources) available. Many scalable peer-assisted content distribution protocols have been proposed, for both the download and streaming contexts.

2.4.1 Peer-assisted Download

Rather than using application-layer multicast, existing scalable downloading techniques, such as BitTorrent [48], allow each peer to download content from *any* peer that has content that it does not have, and do not require any organized delivery structure. These techniques are flexible, and can easily adapt in environments where peer connections are typically heterogeneous with time-varying bandwidths and/or peers frequently join or leave the system. With typical home Internet connections having significantly higher download bandwidth than upload bandwidth (e.g., [156]) peers downloading content in parallel from multiple peers may also better utilize their download bandwidth.

BitTorrent is a popular download protocol for large files that utilize the upload bandwidth of peers to offload the original content source. Files are split into *pieces*, which themselves are split into smaller sub-pieces. Multiple sub-pieces (potentially of the same piece) can be downloaded in parallel from different peers. A peer is said to *have* a piece whenever the entire piece is downloaded. Peers are considered *interested* in all peers that have at least one piece that it currently does not have itself.

The state information about all the peers currently having pieces is maintained by a tracker, while information about the original file and its pieces are stored in a torrent file. Typically, a client wanting to download a file, first obtains the torrent file (e.g., through a webpage), extracts the URL of the tracker (from the torrent file), and contacts the tracker, which replies with a list of peers that have pieces of the file. After connecting to the peers specified in this list, the client finds out which pieces each of these peers have, and starts requesting the pieces that it needs.

BitTorrent distinguishes between peers that have the entire file (called *seeds*), and peers currently downloading the file, that only have parts of the file (called *leechers*). Studies have shown that it is not uncommon for the number of long-lived seeds currently active in the system to be an order of magnitude less than the number of active leechers [91, 111, 132]. To achieve efficient download it is therefore important that leechers contribute to the total upload capacity of the system.

To achieve fairness, load balancing, and high piece diversity a number of ad hoc policies are used in BitTorrent. There are currently many different versions of the BitTorrent client used on the Internet, each with different characteristics. This section only describes the characteristics of the most fundamental policies.[7]

With BitTorrent each peer establishes persistent connections with a large set of peers; however, at each time instance each peer only uploads to a limited number of peers.[8] This is accomplished through a process (called *choking*) in which the peer ceases (or chokes) the upload process to all other peers. Only peers that are unchoked may be sent data. Generally, clients re-evaluate the set of unchoked peers relatively frequently (e.g., every 10 seconds, each time a peer becomes interested/uninterested, and/or each time a new connection is established/broken).

To discourage free-riding, BitTorrent uses a *tit-for-tat* policy in which leechers give upload preference to the leechers that provide the highest download rates. To probe for better pairings (or in the case of the seeds, allows a new peer to download pieces), periodically, typically every third time the set of unchoked peers is re-evaluated, each client uses an *optimistic unchoke* policy, in which a random peer is unckoked.

Without any measure of the upload rates from other peers, seeds were originally proposed to give preference to peers for which a high download rate would be achieved [48]. However, as this can allow peers to monopolize the seed upload bandwidth, it has been found beneficial that seeds always upload to a set of randomly selected peers. To determine new peers to unchoke the seeds always use optimistic unchoking. Further,

[7] Legout *et al*. [110] provide a more detailed description of these policies. Using a recent version of the original BitTorrent client (sometimes called the *mainline* client) they explore the impact these policies have on the performance of a client.

[8] The number of concurrent uploads is client dependent. Whereas the original mainline client used a fixed number of upload connections, newer versions of this client determine the number of upload connections

when determining which peers should remain unchoked, the seeds give preference to recently unchoked peers [110].

With the exception of the first few requested pieces (that are often randomly selected) BitTorrent employs a *rarest-first* policy in which a peer always requests a piece from the set of pieces that are the rarest in the set of all pieces that the peers that it is connected to have, and that it does not have itself. While this policy has been shown to achieve good piece diversity [110, 111], the impact on performance of alternative piece selection policies have not been studied.

Many recent studies have considered the download performance of BitTorrent. It has been shown that this protocol adapts well to changing demands [185], that sharing (using the tit-for-tat policy) is generally done between clients with similar upload bandwidth [134], and that performance does not critically depend on user behavior or piece selection strategies such as rarest-first [120]. Other work has noted that there is an imbalance in the contribution made by different peers, and proposed modifications that try to ensure that peers contribute more evenly [25].

A variety of extensions have been proposed to improve the performance of BitTorrent-like systems [78, 162]. For example, assuming the existence of an origin server, acting as a persistent seed, Slurpie [162] uses a distributed algorithm to organize the downloading peers into a mesh, through which information is propagated. The load at the server is kept independent of the number of peers in the mesh using a distributed probabilistic back-off algorithm, which establishes individual back-off probabilities based on estimates of the number of current downloaders in the mesh and whether each peer is eligible to download from the server.

Erasure coding can be used to improve the efficiency of parallel data retrieval in peer-assisted systems [28, 102, 78]. Rather than exchanging regular pieces, these systems exchange encoded pieces called blocks. While blocks are likely to be useful to more peers than regular pieces, not all blocks are useful to a peer. In particular, there is no benefit retrieving two copies of the same block. To estimate which peers are likely to have useful blocks peers can use techniques such as Bloom filters or approximate

based on its maximum upload rate. Other clients allow the user to explicitly set the number of concurrent upload connections.

reconciliation trees [28]. Network coding can be used to decrease the likelihood of peers exchanging identical (or otherwise less useful) blocks. With distributed network coding, each peer re-encodes the blocks before uploading them to other peers [28, 78]. By re-encoding data throughout the entire network, Gkantsidis *et al*. [78] observe improvements in download times of 20-30%, compared to when encoding is only done at the origin server, and observe improvements of 100-200%, compared to when no encoding is used.

2.4.2 Peer-assisted Streaming

Various peer-to-peer systems have been developed that stream live content using some application-level multicast architecture [95, 125, 85, 39, 102]. However, as with any tree delivery topology, the maximum achievable rate to a node is constrained by the minimum of the rate between any of the upstream peers. Therefore, the transmission rate that any application-level multicast tree can achieve is limited by the monotonically decreasing achievable bandwidth in such a delivery tree.

To efficiently utilize the upload bandwidth of peers with highly diverse upload bandwidths, and resolve the potential bottleneck caused by individual peers with low upload rates, many protocols utilize some form of parallel content delivery [39, 102]. For example, SplitStream [39] splits a content stream into multiple low-rate streams and broadcasts these streams over disjoint multicast trees. This ensures that nodes are not directly limited by the upload rate of any particular peer, and all nodes contribute with upload bandwidth (providing load balancing among peers).

Common to all of the above peer-assisted streaming protocols is that relatively long-duration streams of stable minimum bandwidth must be established. In contrast, but similar to BitTorrent [48], a number of recent peer-assisted systems in which peers actively pull pieces from other peers participating in the same stream have successfully been used to achieve live-streaming of various TV programs and/or events [86, 191, 190, 112]. By exchanging buffer maps, containing information about the pieces that each peer has, peers can request (pull) pieces that they need soon from other peers. With peers being at roughly the same play point, peers typically have a small window of pieces that they exchange among each other. To determine which pieces to request from each peer, Zhang *et al*. [190] suggest a heuristic in which the rare pieces are given preference. If a piece can be provided by multiple peers, preference is given to the peer with the highest available bandwidth.

While most of these systems are proprietary, it appears that these systems do not typically use tit-for-tat, or other incentive mechanisms, instead high-bandwidth peers with high upload bandwidth capacity are expected to upload much more than they download [86]. Peers will request information about additional peers to connect to when not achieving sufficient download rates. By both pulling and pushing pieces increased propagation rates can be achieved [191]. Increased resource utilization and load balancing can be achieved using various overlay maintenance techniques (e.g., taking locality and peer load into account) [112].

Client caches can be used together with tree-delivery structures to achieve on-demand peer-assisted streaming of stored media [163, 53, 24, 161]. Both oStream [53] and OSMOSIS [24] implement streaming using a cache-and-relay strategy, in which each peer typically receives content from a single sender (i.e., its parent). These protocols assume that all peers can retrieve and forward the content at the play rate of the file. After playing a piece of the content, this piece is stored in the client cache, from which it can later be forwarded to client(s) that are at a later play point of the file. The authors of dPam [161] observe that peers are often capable of downloading at a higher rate than the play rate and suggest using a pre-fetch-and-relay technique. The pre-fetched data allows the peers to better handle departures of upstream peers.

Other work considers systems in which each peer connects to multiple senders (i.e., some set of servers and/or peers) from which data is streamed sequentially; however, in contrast to systems such as SplitStream [39] the data and rates from each sender is dynamically adjusted by the receiver [143, 118, 130, 85]. Piotrowski *et al.* [130] use a piece selection algorithm in which pieces are requested sequentially from individual senders. If the achieved download rate from the sender does not allow a piece to be downloaded by its expected playout time, the client attempts to increase the rate the piece is downloaded, by either re-assigning which pieces are downloaded from each sender, or splitting the piece into sub-pieces. Rejaie *et al.* [143, 118] assume that the data is layered (using layered or multiple description encoding) and propose techniques for the client to adjust its streaming quality. To accommodate for time-varying download rates they suggest that each peer requests the pieces corresponding to the layers that best match its current download rate. These protocols do not consider incentive mechanisms (such as the tit-for-tat policy) and sequential piece selection (used by these protocols) can result in poor performance in highly dynamic environments as shown by the results presented in Chapter 5.

In the work in this area most closely related to the work presented in this thesis, Annapureddy *et al*. [16] propose a video-on-demand system for stored files in which each file is split into sub-files. Each sub-file consists of multiple pieces and can be downloaded using a BitTorrent-like approach; sub-files are downloaded sequentially. To improve performance, they suggest pre-fetching a small amount of data from the sub-file that will be required next, and having each peer re-encode data on a sub-file-by-sub-file basis (using distributed network coding). While re-encoding blocks increases the usability of a block, encoding requires that enough blocks of each sub-file are retrieved (and decoded) before the sub-file can be played out. Note that use of large sub-files results in large startup delays, while using very small sub-files results in close to sequential piece selection, which again can lead to poor performance. The best choice of sub-file sizes would be workload (and possibly also client) dependent, although the method requires these sizes to be statically determined. The authors do not elaborate on how the sizes of the sub-files can be chosen, or how startup delays can be dynamically determined.

Chapter 3
Scalable Download from a Single Server

As discussed in Section 2.2, existing multicast-based approaches to scalable download from a single server include *batching* [66, 182] and *cyclic multicast* [146, 10, 26, 176, 31, 150, 27]. In contrast to the work on scalable download using batching, this chapter considers delivery of large files, for which joining an on-going multicast, rather than waiting until the beginning of the next multicast, may provide a significant performance benefit. Also, rather than considering the problem of scheduling delivery of a fixed collection of objects on given channel(s), this chapter considers contexts in which the server bandwidth devoted to this task is somewhat elastic, and thus focus on the *average* bandwidth used for delivery of each file, and the resulting average or maximum client delay. In contrast to this prior work on cyclic multicast, this chapter focuses on the performance comparison between batching and cyclic multicast, and the design of hybrid protocols that combine elements of both approaches to achieve superior performance.

This chapter considers the problem of devising protocols that minimize the average or maximum client delay for downloading a single file, as a function of the average server bandwidth used for delivery of that file. An equivalent problem is to minimize the average server bandwidth required to achieve a given average or maximum client delay. This equivalent perspective is sometimes adopted. Although delivery of multiple files is not explicitly considered, note that use of a download protocol that minimizes the average server bandwidth for delivery of each file will minimize the average total required server bandwidth for delivering all files, as well.

Focusing first on systems with homogeneous clients that have identical reception rate constraints lower bounds are developed on the average and maximum client delay for downloading a file, as functions of the average server bandwidth used for delivering that file. It is found that neither batching nor cyclic multicast consistently yields delays close to optimal. Motivated by these results, new practical protocols are developed that largely close these gaps. The new protocols achieve within 15% of the optimal maximum delay and 20% of the optimal average delay, in homogeneous systems.

Next, protocols for delivery of a file to heterogeneous clients that have widely varying achievable reception rates are considered. In this context, achieving efficient delivery as well as lower delay for higher rate clients requires use of multiple multicast channels. Each client listens to the number of channels corresponding to its achievable reception rate. The key challenge is to achieve a close-to-optimal compromise between high-rate transmissions (in aggregate, over all channels used for a file), which enable lower delays for clients that can receive at such rates, and low-rate transmissions that allow maximal sharing. A protocol for delivery to heterogeneous clients is proposed that yields maximum client delays that are within 25% of optimal in the scenarios considered.

The remainder of the chapter is organized as follows. Section 3.1 defines and analyzes the basic batching and cyclic multicast protocols. In this section, as in the subsequent two sections, homogeneous clients are assumed. Lower bounds on the average and maximum client delay for downloading a single file, for given average server bandwidth usage (or, equivalently, on the average server bandwidth required to achieve a given average or maximum client delay) are derived in Section 3.2. Section 3.3 develops new scalable download protocols that achieve close to optimal performance. Protocols for delivery to heterogeneous clients are developed and evaluated in Section 3.4. Summary and conclusions are presented in Section 3.5.

3.1 Baseline Policies

This section defines and analyzes simple “baseline” batching and cyclic multicast protocols for delivery of a single file, assuming homogeneous clients. The metrics of interest are the average client delay (i.e., download time), the maximum client delay in cases where such a maximum exists, and the average server bandwidth used for the file data multicasts. It is assumed throughout the chapter that each requesting client receives the entire file; i.e., clients never balk while waiting for service to begin or after having received only a portion of the file. The analysis and protocols presented are compatible with erasure-coded data. Each client is assumed to have successfully received the file once it has listened to multicasts of an amount of data L (termed the "file size" in the following, although with packet loss and erasure coding, L may exceed the true file size). Poisson request arrivals are assumed unless otherwise specified. Generalizations are discussed in some cases. Note that Poisson arrivals can be expected for independent requests from large numbers of clients. Furthermore, multicast delivery protocols that have high performance for Poisson arrivals, have even better performance under the more bursty arrival processes that are typically found in contexts where client requests are not independent [68].

3.1.1 Batching

Consider first batching protocols in which the server periodically multicasts the file to those clients that have requested it since it was last multicast. Any client whose request arrives while a multicast is in progress, simply waits until the next multicast begins.

Perhaps the simplest batching protocol is to begin a new multicast of the file every t time units for some constant t. However, this protocol has the disadvantage that multicasts may sometimes serve no or only a few clients.

Two optimized batching protocols are considered here. The first, termed *batching/constant batching delay* (*batching*/*cbd*), achieves the minimum average server bandwidth for a given maximum client delay, or equivalently the minimum value of maximum client delay for a given average server bandwidth, over the class of batching protocols as defined above. Letting T denote the time at which some file multicast begins and a denote the duration of the time interval from T until the next request arrival, the server will begin the next multicast at time T+a+Δ, where Δ is a parameter of the protocol. Thus, using the notation defined in Table 3.1, the average time between file multicasts is Δ+1/λ, the average server bandwidth B is L/(Δ+1/λ), and the maximum client delay D is Δ plus L/r (the file transmission time). Here, λ is the rate at which the file is requested and the transmission rate r on the multicast channel is at most equal to the maximum sustainable client reception rate b. With respect to the average client delay A, note that the client whose request arrival triggers the scheduling of a new multicast experiences the maximum waiting time Δ until the multicast begins. All

Table 3.1: Notation used in Chapter 3.

Symbol	**Definition**
λ	File request rate
L	File size
b	Maximum sustainable client reception rate
r	Transmission rate on a multicast channel ($r \le b$)
B	Average server bandwidth
A	Average client delay (time from file request, until file is completely received)
D	Maximum client delay
Δ	Batching delay parameter (threshold value on the maximum time until service)
n	Batching delay parameter (threshold value on the number of requests)
f	Batching delay parameter (fraction of times n+1 should be used as a threshold, rather than n)

clients whose requests arrive during the batching delay Δ will share reception of this multicast. On average, there will be $\lambda\Delta$ such clients, and the average waiting time until the multicast begins for such a client will be $\Delta/2$. In summary, [9]

$$B_{b/cbd} = \frac{L}{\Delta + 1/\lambda}; \tag{3.1}$$

$$A_{b/cbd} = \frac{\Delta(1+\lambda\Delta/2)}{1+\lambda\Delta} + L/r; \qquad D_{b/cbd} = \Delta + L/r\,. \tag{3.2}$$

The second optimized batching protocol, termed *batching/request-based delay* (*batching/rbd*), achieves the minimum value of average client delay for a given average server bandwidth, over the class of batching protocols as defined above.[10] The basic idea is to make the batching delay some integral number of request inter-arrival times. To make it possible to achieve arbitrary average server bandwidth values, the protocol is defined such that the server waits for n+1 requests for a fraction f of its multicasts, and for n requests for the remaining fraction 1–f, where n and f are protocol parameters

[9] In the non-Poisson case, assuming request interarrival times are independent and identically distributed (IID), these performance metrics can be obtained by calculating conditional expectations. For example, note that $1/\lambda$ in the bandwidth expression can be replaced with the expected time from after the initiation of a transmission until the next request, conditioned on the fact that there was a request arrival time Δ in the past.

[10] This can be established formally using an argument similar to that used for the lower bound on average server bandwidth in Section 3.2.1.

(integer $n \geq 1$, $0 \leq f < 1$).[11] Thus, the average time between file multicasts is $(n+f)/\lambda$, and the average server bandwidth is $L/((n+f)/\lambda)$. The average client delay can be derived from the fact that each multicast serves n clients plus with probability f one additional client, and the i'th last of these clients experiences an average waiting time until the multicast begins of $(i-1)/\lambda$. Note that the maximum client delay is unbounded with this protocol. Thus,

$$B_{b/rbd} = \frac{L}{(n+f)/\lambda}; \tag{3.3}$$

$$A_{b/rbd} = \frac{\frac{n(n-1)}{2\lambda} + f\frac{n}{\lambda}}{n+f} + L/r = \frac{n(n+2f-1)}{2\lambda(n+f)} + L/r; \quad D_{b/rbd} \text{ is unbounded.} \tag{3.4}$$

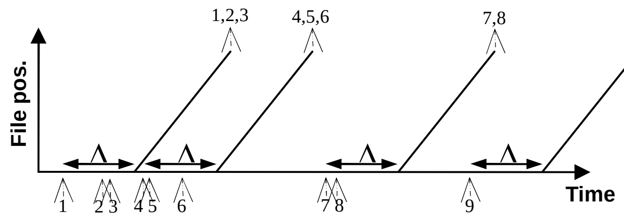
Note that for both of these batching protocols, the value of the multicast transmission rate r that minimizes average and maximum client delay is equal to the maximum sustainable client reception rate b.

Figure 3.1 illustrates the operations of these two batching protocols, as well as the cyclic multicast protocol discussed in the next section, for an example sequence of requests. Requests are numbered and the arrival times and service completion times of the requests are indicated by the arrows at the bottom and top of each subfigure, respectively. The solid, slanted lines denote multicast transmissions, each of which, in the case of the batching protocols, delivers the entire file. For the *batching/cbd* protocol, the batching delays (each of duration Δ) are indicated with double arrows along the horizontal (time) axis.

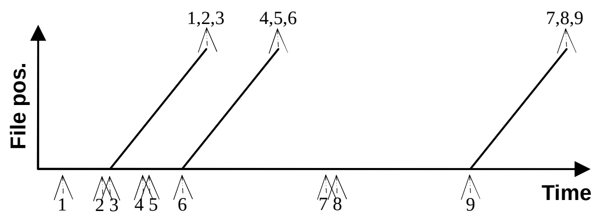
3.1.2 Cyclic Multicast

Perhaps the simplest cyclic multicast protocol is to continually multicast file data at a fixed rate r (cycling back to the beginning of the file when the end is reached) on a single multicast channel, regardless of whether or not there are any clients listening. Instead, consider a more efficient cyclic multicast protocol, *cyclic/listeners (cyclic/l)*, that assumes that the server can determine whether there is at least one client with an unfulfilled request for the file, and transmit only if there is. Since the server transmits

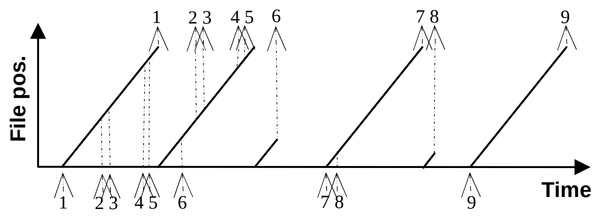
[11] When arrivals are Poisson, inter-arrival times are memoryless, and the method by which the server determines when to wait for n versus n+1 arrivals (for fixed f) has no impact on average server bandwidth usage or average delay.



(a) Batching/constant batching delay



(b) Batching/request-based delay (n=3, f=0)



(c) Cyclic/listeners

Figure 3.1: Operation of the Baseline Protocols for an Example Request Sequence.

whenever there is at least one client, the delay experienced by each client is just the file transmission time, L/r. The average server bandwidth can be derived by noting that there will be at least one client listening on the multicast channel at an arbitrary point in time T, if and only if at least one request for the file was made during the time interval $[T–L/r, T]$, and that the probability of at least one request arrival during an interval of duration L/r is $1-e^{-\lambda L/r}$ for Poisson arrivals at rate λ.[12] This yields

$$B_{c/l} = r\left(1-e^{-\lambda L/r}\right); \tag{3.5}$$

$$A_{c/l} = D_{c/l} = L/r\ . \tag{3.6}$$

Note that the transmission rate r is the only protocol parameter, and by itself determines the tradeoff between server bandwidth usage, and client delay.

[12] Note that the performance of this protocol can be analyzed for any arrival process for which it is possible to compute the probability of there being at least one request arrival during a randomly chosen time period of duration L/r.

3.2 Lower Bounds

Making the same assumptions as in Section 3.1 of homogeneous clients, full-file delivery, and Poisson client request arrivals, this section derives fundamental performance limits for scalable download protocols. These limits depend on the maximum sustainable client reception rate. Note that for batching protocols, for example, if the server transmission rate is increased the batching delay can be increased without increasing the total client delay, thus providing a longer period over which aggregation of requests can occur and more efficient use of server bandwidth. Section 3.2.1 considers the limiting case in which clients can receive data at arbitrarily high-rate, for which there is a previously derived bound on maximum delay [171]. Section 3.2.2 considers the realistic case in which there is an upper bound *b* on client reception rate.

3.2.1 Unconstrained Client Reception Rates

Consider first the maximum client delay, and the average server bandwidth required to achieve that delay. From Tan *et al.* [171], [13]

$$B \geq \frac{L}{D+1/\lambda} \iff D \geq L/B - 1/\lambda \,. \tag{3.7}$$

This bound is achieved in the limit, as the server transmission rate tends to infinity, by a protocol in which the server multicasts the file to all waiting clients whenever the waiting time of the client that has been waiting the longest reaches *D*.

Consider now the problem of optimizing for average client delay. At each point in time an optimal protocol able to transmit at infinite rate would either not transmit any data, or would transmit the entire file. To see this, suppose that some portion of the file is transmitted at an earlier point in time than the remainder of the file. Since client requests might arrive between when the first portion of the file is transmitted and when the remainder is transmitted, it would be more efficient to wait and transmit the first portion at the same time as the remainder. Optimizing for average client delay requires determining the spacings between infinite rate full file transmissions that are optimal for this metric. With Poisson arrivals and an on-line optimal protocol, (1) file transmissions

[13] As with the bandwidth expression for *batching/cbd* in Section 3.1, for the case of non-Poisson request arrivals with IID request interarrival times the $1/\lambda$ term can be replaced by the appropriate conditional expectation. Further note that a bandwidth lower bound can be obtained for any process such that this quantity can be bounded from above, as has been noted in the scalable streaming context [68].

occur only on request arrivals, and (2) each multicast must serve either n or n+1 clients for some integer $n \geq 1$. With respect to this latter property, consider a scenario in which the file is multicast to n waiting clients on one occasion and to n+k clients for $k \geq 2$ on another. A lower average delay could be achieved, with the same average spacing between transmissions, by delaying the first multicast until there are n+1 waiting clients, and making the second multicast at the request arrival instant of the n+k–1th client instead of the n+k^{th}.

Thus, a lower bound on the average server bandwidth B required to achieve a given average client delay A can be derived by finding an integer $n \geq 1$, and value f ($0 \leq f < 1$), such that

$$A = \frac{\frac{n(n-1)}{2\lambda} + f\frac{n}{\lambda}}{n+f} = \frac{n(n+2f-1)}{2\lambda(n+f)}, \tag{3.8}$$

in which case

$$B \geq \frac{L}{(n+f)/\lambda}. \tag{3.9}$$

Equivalently, to determine a lower bound on the average delay A that can be achieved with average server bandwidth B, let $n = \max[1, \lfloor \lambda L/B \rfloor]$, and $f = \max[0, \lambda L/B - n]$. Then,

$$A \geq \frac{n(n+2f-1)}{2\lambda(n+f)}. \tag{3.10}$$

Note that for $B < \lambda L$ (the bandwidth required for unicast delivery), the optimal protocols for minimizing the average delay A and the maximum delay D are different, and thus the lower bounds on A and D cannot be achieved simultaneously. In fact, for all $B < \lambda L$ the optimal protocol for average delay has unbounded maximum delay. If $\lambda L/B$ is an integer greater than one, the lower bound on A is exactly half the lower bound on D; otherwise, it is somewhat greater than half. In particular, as B tends to λL, the ratio of the lower bounds on A and D tends to one.

3.2.2 Constrained Client Reception Rates

Assume now that clients have a finite maximum sustainable reception rate b. In this case, both the maximum and average delay must be at least L/b. To achieve the minimal values $D = A = L/b$, each client must receive the file at maximum rate starting immediately upon its request. The *cyclic/l* protocol defined in Section 3.1.2 achieves the lowest possible server bandwidth usage in this case, as the transmission rate of the server is (only) b whenever there is at least one active client, and zero otherwise. Thus, for $D = A = L/b$, the bound becomes $B \geq (1 - e^{-\lambda L/b})$.

More generally, for a specified maximum delay $D \geq L/b$, the average server bandwidth is minimized by the *send as late as possible* (*slp*) protocol, in which the server cyclically multicasts file data at rate b whenever there is at least one active client that has no "slack" (i.e., for which transmission can no longer be postponed). Such a client must receive data continuously at rate b until it has received the entire file, if it is to avoid exceeding the delay bound. Note that although this protocol is optimal for maximum delay, it requires that the server maintain information on the remaining service requirements and request completion times of all outstanding requests. Furthermore, the *slp* protocol can result in extremely fragmented transmission schedules. This motivates simpler and more practical near-optimal protocols such as that devised in Section 3.3.1.

An accurate approximation for the average server bandwidth with the *slp* protocol is given by

$$B_{slp} \approx \left(\frac{(e^{\lambda L/b} - 1)/\lambda + D - L/b}{e^{\lambda L/b}/\lambda + D - L/b} \right) \frac{L}{D}. \tag{3.11}$$

Here the L/D factor approximates the average server bandwidth usage over those periods of time during which there is at least one active client (i.e., client with an outstanding request). The factor in brackets approximates the fraction of time that this condition holds. This fraction is equal to the average duration of a period during which there is at least one active client, divided by the sum of this average duration and the average request inter-arrival time ($1/\lambda$). The average duration of a period during which there is at least one active client is approximated by the average duration of an $M/G/\infty$ busy period with arrival rate λ and service time L/b, as given by $(e^{\lambda L/b} - 1)/\lambda$[14], plus the duration of the delay after the arrival of a request to a system with no active clients until

[14] This expression can be derived by observing that the probability that the system is idle (i.e., $e^{-\lambda L/b}$) is equal to the expected duration of an idle period (i.e., $1/\lambda$) divided by the expected duration of a full cycle

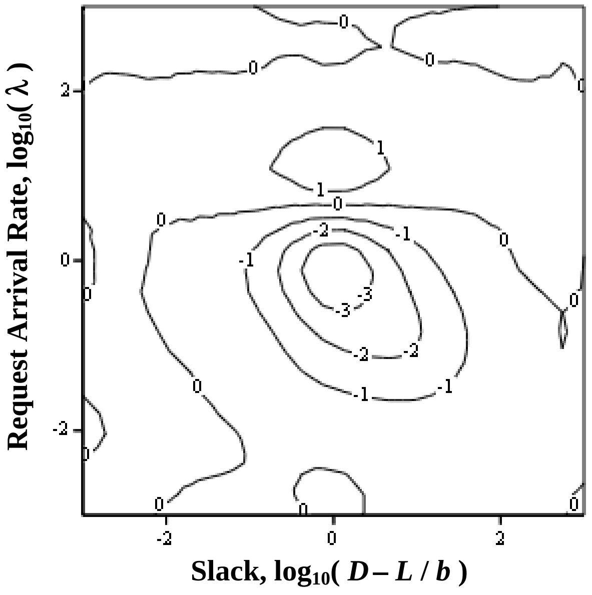


Figure 3.2: Lower Bound Approximation (% relative error contours; unit of data volume is the file, unit of time is the time required to download the file at maximum rate: i.e., L = 1, b = 1).

the server must begin transmitting (D–L/b). Note that a corresponding approximation for the minimum achievable maximum delay, for given average server bandwidth, can be obtained by solving for D in the above approximation.

Exhaustive comparisons against simulation results indicate that the above approximation is very accurate, with relative errors under 4%, and thus the remainder of the chapter uses the approximation rather than simulation values.[15] Figure 3.2 summarizes the validation results, showing contours of equal error over a two dimensional space. Negative and positive errors correspond to underestimations and overestimations of the true values as obtained from simulation, respectively. Without loss of generality, the unit of data volume is chosen to be the file, and the unit of time is chosen to be the time required to download the file at the maximum sustainable client reception rate. With these choices of units, L and b are each equal to one. The only two remaining parameters are λ and D. The logarithm of the arrival rate λ is used on the

including both an idle period (of expected duration 1/λ) and a busy period (the expected duration of which can be solved for).

[15] All simulations make the same system and workload assumptions as the analytic models (including the assumption of Poisson arrivals). Note that where both simulation and analytic results are presented, the

vertical axis of the contour plot, covering six orders of magnitude of arrival rates, while six orders of magnitude of "slack" are covered on the horizontal axis using the logarithm of $D–L/b$. As can be seen directly from the approximation, this expression is exact for the boundary cases of $\lambda \to 0$ (minimum λ), $\lambda \to \infty$ (maximum λ), $D \to \infty$ (maximum D), $L \to 0$ (minimum L), $b \to \infty$ (maximum b), and $D = L/b$ (minimum D, or maximum L, or minimum b), holding the other parameters fixed in each case. For example, note that for $b \to \infty$ the approximation reduces to $L/(D+1/\lambda)$, and for $D = L/b$ the approximation reduces to $b(1-e^{-\lambda L/b})$.

The optimal scalable download protocol for *average* delay, under a reception rate constraint, appears to be very difficult to determine in general. However, a lower bound can be derived as follows. As noted previously, for $A = L/b$ the optimal protocol is *cyclic/l* as defined in Section 3.1.2, with $r = b$. Furthermore, a variant of cyclic multicast in which the server sometimes or always waits until a second request arrival before beginning transmission will also be optimal, for values of average delay and bandwidth that can be achieved by this protocol, since each unit of additional channel idle time is achieved by delaying the minimum possible number of clients (only one). Letting f denote the fraction of idle periods in which channel transmission does not begin until a second request arrives, the server bandwidth and average delay under this *cyclic/wait for second, listeners* (*cyclic/w*2,*l*) protocol are given by

$$B_{c/w2,l} = b\frac{\left(e^{\lambda L/b}-1\right)/\lambda}{\left(e^{\lambda L/b}-1\right)/\lambda+\left(1+f\right)/\lambda} = b\frac{e^{\lambda L/b}-1}{e^{\lambda L/b}+f}; \tag{3.12}$$

$$A_{c/w2,l} = \frac{f/\lambda}{\lambda\left((e^{\lambda L/b}-1)/\lambda+(1+f)/\lambda\right)}+L/b = \frac{f/\lambda}{e^{\lambda L/b}+f}+L/b. \tag{3.13}$$

Note here that $(e^{\lambda L/b}-1)/\lambda$ is the average duration of an $M/G/\infty$ busy period with arrival rate λ and service time L/b, and $(1+f)/\lambda$ is the average duration of a channel idle period. For server bandwidth values B that can be achieved with this protocol, it can be shown (by solving for f in terms of B and then substituting into the average delay expression) that,

purpose of the simulation is to assess the accuracy of the approximations made in the analysis, and not for validation of the system or workload assumptions.

$$A \geq \max\left[0, \frac{\left(e^{\lambda L/b} - 1\right)b/B - e^{\lambda L/b}}{\lambda\left(e^{\lambda L/b} - 1\right)b/B}\right] + L/b\ . \tag{3.14}$$

Equivalently, to determine the lower bound on the average server bandwidth *B* that can be achieved with average delay *A*, solving for *f* in terms of *A* and substituting into the average server bandwidth equation yields

$$B \geq \max\left[0, b\left(1 - e^{-\lambda L/b}\right)\frac{1/\lambda - (A - L/b)}{1/\lambda}\right]. \tag{3.15}$$

Values of *B* that are smaller (or values of *A* that are larger) than those achieved for *f* = 1 are not achievable by the *cyclic/w*2,*l* protocol, because in this protocol each idle period always ends no later than the time of the second request arrival. However, the above bounds are valid (although unachievable) for those smaller values of *B* (and larger values of *A*) that can be obtained by substituting values greater than one for the parameter *f* in the above expressions. The bounds are valid in this case because even for *f* > 1, these expressions still assume that the minimum number of clients is delayed (i.e., only one) before the server begins transmission. The bounds are unachievable since the average duration of this delay is assumed to be *f*/λ, which for *f* > 1 is greater than the average delay until the second request arrival.

A second lower bound on average delay can be derived as follows. First, note that in an optimal protocol, data transmission will always occur at rate *b*, since: (1) each client can receive at rate at most *b*, and (2) the average delay cannot increase when a period of length *l* between request completions during which the transmission rate is less than *b*, is replaced by an idle period followed by a period of transmission at rate *b* (of combined length *l* and equal total server bandwidth usage).

Suppose now that each request arrival that occurs during a busy period is shifted earlier, so that it occurs at a multiple (possibly zero) of *L*/(2*b*) from the start of the busy period. As a result of this shifting, requests arriving during a busy period will have greater likelihood of completing service before the busy period ends, for a fixed busy period duration. Therefore, average delay cannot increase. It is now possible to determine the optimal protocol, assuming this shift of request arrivals, based on the following three observations: (1) by the same arguments as in Section 3.2.1, in the optimal protocol each idle period must end once *n*, or *n*+1 with some probability *f*, requests have been accumulated, for some integer $n \geq 1$ and $0 \leq f < 1$; (2) each busy period must end on a request completion, and therefore in the optimal protocol be of total length equal to a multiple (at least two) of $L/(2b)$; and (3) since the state of the system at each multiple of $L/(2b)$ within a busy period is entirely captured by the number of request arrivals that occurred within the previous $L/(2b)$ (all of whose respective clients have been listening to the channel for exactly time $L/(2b)$, owing to the shifting), there is an integer threshold $k \geq 1$ such that if the number of such arrivals is less than k, the server will stop transmitting in the optimal protocol (thus ending the busy period), and otherwise it will not. Note that these observations uniquely specify the operation of the optimal protocol, by establishing the criteria used for determining when to start a transmission, specifying the possible instances when a transmission can be completed, and for each of these time instances specifying the criteria used to determine if the transmission should be stopped.

Given values for the parameters n, f, and k, the average server bandwidth and the average client delay with this (unrealizable) *shifted arrivals* (*sa*) protocol are given by

$$B_{sa} = b \frac{(1+1/p)L/(2b)}{(1+1/p)L/(2b) + \left(n + f - \sum_{i=0}^{k-1} i \frac{p_i}{p} \right) \Big/ \lambda}, \tag{3.16}$$

$$A_{sa} = \frac{L}{b} + \frac{\sum_{i=0}^{k-1} \frac{p_i}{p} \left[i(n-i)\frac{1}{\lambda} + \frac{(n-i)(n-i-1)}{2\lambda} + \frac{n}{\lambda} f \right]}{\lambda \left((1+1/p)L/(2b) + \left(n + f - \sum_{i=0}^{k-1} i \frac{p_i}{p} \right) \Big/ \lambda \right)}, \tag{3.17}$$

where $p_i = \frac{1}{i!}(\lambda L/(2b))^i e^{-\lambda L/(2b)}$ is the probability of i request arrivals in time $L/(2b)$, and $p = \sum_{i=0}^{k-1} p_i$ is the probability of a busy period ending when its duration reaches a multiple of $L/(2b)$ (and at least L/b). B_{sa} is given by the ratio of the average duration of a busy period to the sum of the average durations of a busy period and an idle period, times the transmission rate b. Note here that when the busy period ends owing to having $i < k$ request arrivals during the previous $L/(2b)$, the average duration of the idle period will be $(n+f-i)/\lambda$, since only $n-i$ (or $n+1-i$) new requests need be received to obtain a total of n (or $n+1$) unsatisfied requests. A_{sa} is equal to the total expected idle time incurred by those clients making requests during a busy period and the following idle

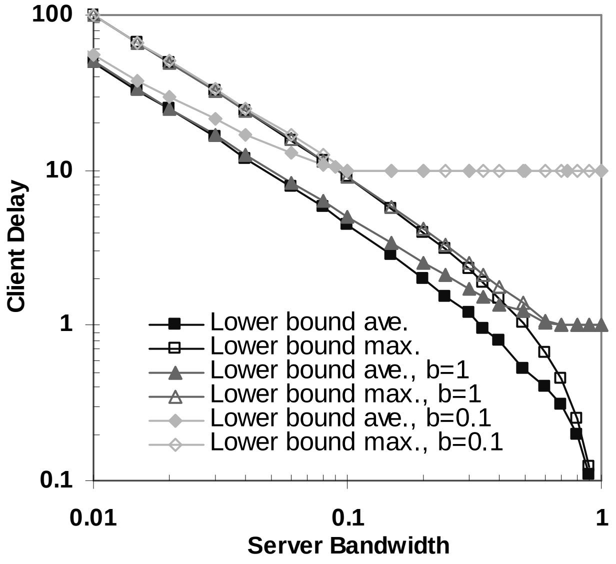


Figure 3.3: Lower Bounds on Client Delay (unit of data volume is the file, unit of time is the average time between requests: i.e., $L = 1$, $\lambda = 1$).

period, divided by the expected number of such requests, plus the time required to download the file data (L/b). The optimal n, f, and k values for a particular server bandwidth or average client delay can be found numerically, so as to obtain a lower bound on average delay or server bandwidth, respectively. This bound can then be combined with the corresponding bound from the *cyclic/w*2,*l* protocol analysis, to yield a single lower bound, by taking the maximum of the two.

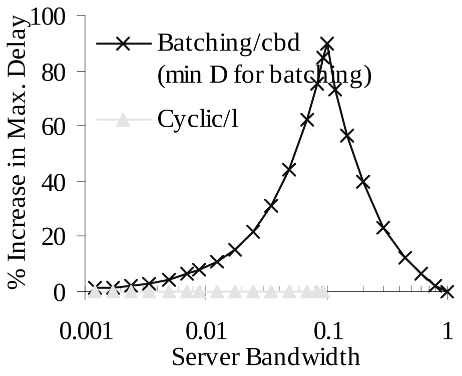
3.2.3 Lower Bound Comparisons

Figure 3.3 shows the lower bounds on average and maximum client delay for the case of unconstrained client reception rates and for $b = 1$ and $b = 0.1$. Without loss of generality, the unit of data volume is chosen to be the file and the unit of time is chosen to be the average time between requests. With these choices of units, $L = 1$, $\lambda = 1$, client delay is expressed as a normalized value in units of the average time between requests, average server bandwidth is expressed as a normalized value in units of file transmissions per average time between requests, and the maximum sustainable client reception rate is expressed as a normalized value in units of file receptions per average time between requests. These units are used in all figures comparing homogenous client protocols (Sections 3.2 and 3.3). Note that the average server bandwidth B in these units can be interpreted as the fraction of the average bandwidth required for unicast delivery, so the region of interest in the design of scalable multicast protocols corresponds to values of B considerably less than one.

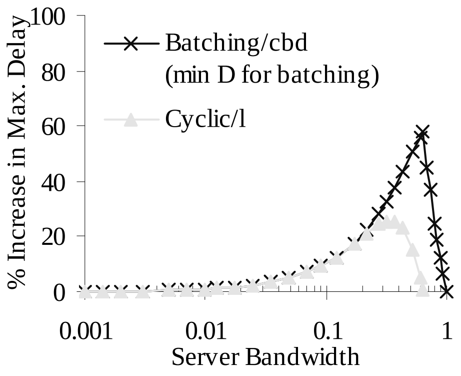
Although the above choice of data volume and time units correctly reflects the fact that it is server bandwidth and client reception rate *relative* to request rate and file size that determines performance, some care is required in interpreting the resulting figures. Consider, for example, Figure 3.3, and a scenario in which the client request rate decreases for fixed average server bandwidth (when expressed in unnormalized units). With the chosen units λ remains equal to one in this scenario (since the unit of time is the average time between requests), but B (expressed in units of file transmissions per average time between requests) increases proportionally to the decrease in the client request rate. Thus, in Figure 3.3, the increasing value of the normalized server bandwidth B as one moves from left to right on the horizontal axis can correspond to increasing server bandwidth (with a fixed client request rate) or decreasing client request rate (with a fixed server bandwidth). Similar considerations apply with respect to the normalized maximum sustainable client reception rate b.

Perhaps the main observation from Figure 3.3 is that client reception rate constraints can strongly impact the achievable performance, although this impact diminishes as the value of the normalized average server bandwidth B decreases. Note also that the difference between the average and maximum delay bounds decreases with increasing server bandwidth. The point where these bounds become identical is the point at which each client experiences only the minimum delay of L/b.

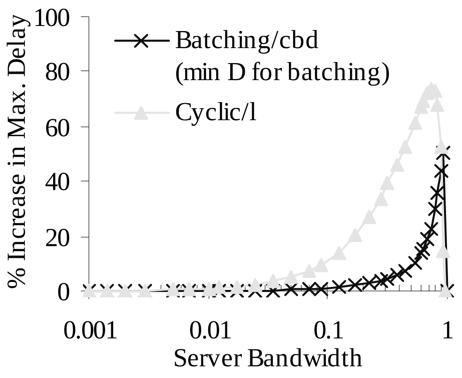
Figure 3.4 plots the percentage increases in the maximum client delay for the baseline batching and cyclic multicast protocols in comparison to the lower bound, for three different values of client reception rate. Figure 3.5 plots the corresponding percentage increases in average client delay for the baseline protocols. The system measures are expressed in the same normalized units as in Figure 3.3. Note that the average server bandwidth with *cyclic/l* cannot exceed b times the fraction of time that there is at least one active client, and thus the rightmost point of each *cyclic/l* curve is for server bandwidth of less than one.



(a) b = 0.1



(b) b = 1



(c) b = 10

Figure 3.4: Maximum Delay with Baseline Protocols Relative to Lower Bound (L = 1, λ = 1).

Figures 3.4 and 3.5 show that the batching protocols are close to optimal for small (normalized) server bandwidths, when many requests are accumulated before the next transmission takes place, and for server bandwidths approaching one, when most clients are served individually with minimal delay of L/b. Batching can be significantly suboptimal for intermediate server bandwidth values, however, particularly for maximum client delay (for example, in Figure 3.4(a), b = 0.1 and B between 0.05 and 0.2). Note also that the overall relative performance of batching degrades as the maximum sustainable client reception rate decreases, since in this case the required duration of a multicast increases, and with the batching protocols new clients are not able to begin listening to a multicast after it has commenced.

In contrast, the performance of *cyclic/l* improves for decreasing client reception rate. However, *cyclic/l* is substantially suboptimal for average client delay over most of

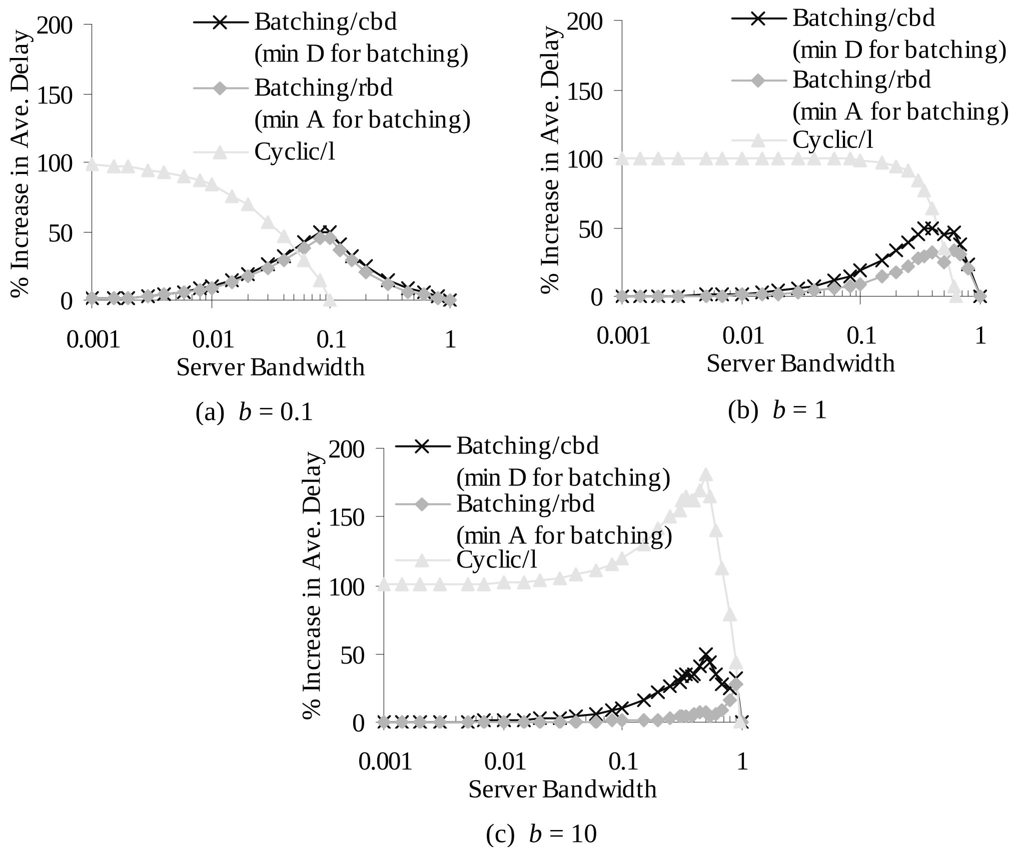


Figure 3.5: Average Delay with Baseline Protocols Relative to Lower Bound (L = 1, λ = 1).

the parameter space, and for maximum delay when the client reception rate is high and the server bandwidth is also high although not approaching one (i.e., in Figure 3.4(c), b = 10.0 and B between 0.4 and 0.9). Note that for small and intermediate server bandwidths, *cyclic/l* is close to optimal for maximum client delay, but since the optimal average client delay is approximately half the optimal maximum client delay in this case, the average client delay with *cyclic/l* is about 100% higher than optimal.

3.3 Near-optimal Protocols

Figures 3.4 and 3.5 suggest that there is substantial room for improvement over the baseline batching and cyclic multicast protocols, since for each of maximum and average client delay there is a region of the parameter space over which each protocol is substantially suboptimal. The main weakness of the batching protocols is that clients that make requests while a multicast is already in progress do not listen to this multicast. All clients receive the file data “in-order”, waiting until the beginning of the next multicast before beginning their downloads. With the baseline cyclic multicast protocol, on the other hand, clients can begin receiving data at arbitrary points in time within an on-going multicast. Since the server transmits whenever there is at least one active client, however, there will be periods over which transmissions serve relatively few clients.

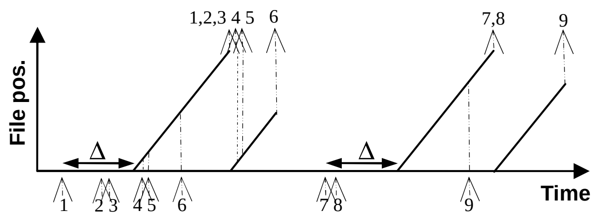
Clearly, an improved protocol should allow clients to begin listening to an on-going multicast at the times of their requests, but should also allow server transmissions to be delayed so as to increase the actual or expected number of clients each serves. It is straightforward to apply a batching-like rule for deciding when a cyclic multicast transmission should commence; the key to devising a near-optimal protocol is determining the conditions under which a multicast should be continued, or terminated. Section 3.3.1 develops and analyzes new protocols that focus on improving maximum client delay, while Section 3.3.2 develops and analyzes protocols whose focus is improved average client delay. As in Sections 3.1 and 3.2, these sections assume homogeneous clients, full-file delivery, and Poisson arrivals. Section 3.3.3 relaxes the Poisson assumption, and considers the worst-case performance of the protocols under arbitrary arrival patterns.

3.3.1 Protocols Minimizing Maximum Delay

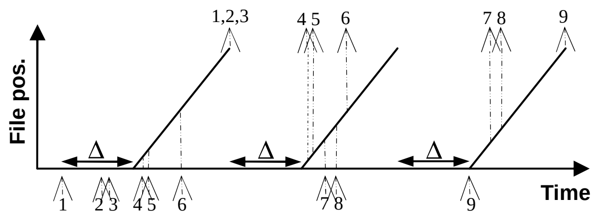
Consider first a simple hybrid of batching and cyclic multicast termed *cyclic/constant delay, listeners* (*cyclic/cd,l*), in which a cyclic multicast is initiated only after a batching delay (as in the *batching/cbd* protocol from Section 3.1.1), and is terminated when there are no remaining clients with outstanding requests (as in the *cyclic/l* protocol). With batching delay parameter Δ and transmission rate r ($r \le b$), the average duration of a channel busy period is given by $(e^{\lambda L/r}-1)/\lambda$, and the average duration of an idle period is given by $1/\lambda+\Delta$. This yields

$$B_{c/cd,l} = r\frac{e^{\lambda L/r}-1}{e^{\lambda L/r}+\lambda\Delta}; \tag{3.18}$$

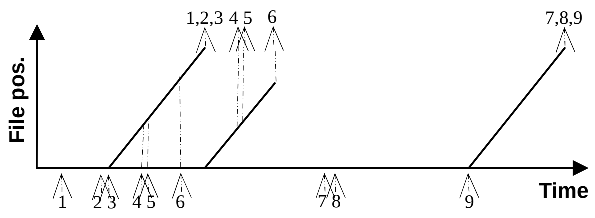
$$A_{c/cd,l} = \frac{\Delta(1+\lambda\Delta/2)}{e^{\lambda L/r}+\lambda\Delta}+L/r\,; \qquad D_{c/cd,l} = \Delta + L/r\,. \tag{3.19}$$



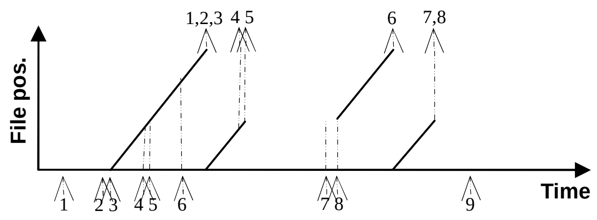
(a) Cyclic/cd,l



(b) Cyclic/cd,bot



(c) Cyclic/rbd,l (n=3, f=0)



(d) Cyclic/rbd,cot (n=3, f=0)

Figure 3.6: Examples Scenarios for Improved Protocols.

The operation of the *cyclic/cd,l* protocol, as well as that of the other protocols developed in this section, is illustrated in Figure 3.6 for the same example pattern of request arrivals as in Figure 3.1.

For $D_{c/cd,l} > L/b$, there are multiple combinations of Δ and r that yield the same maximum client delay. Optimal settings that minimize server bandwidth can be found numerically. Interestingly, $r = b$ is often not optimal. Since a cyclic multicast is continued as long as there is at least one listening client, channel busy periods may have long durations. Under such conditions, it may be possible to reduce server bandwidth usage while keeping the maximum delay fixed by reducing both r and Δ. In particular, note that for $\lambda \to \infty$, the channel is always busy, and thus the optimal r is the minimum possible r (the file size L divided by the maximum delay) and the optimum Δ is zero.

A better hybrid protocol, termed here *cyclic/constant delay, bounded on-time (cyclic/cd,bot)*, can be devised by using a better policy for when to stop transmitting. The key observation is that the duration of a multicast transmission can be limited to at most L/r without impact on the maximum client delay. As in the *cyclic/cd,l* protocol, a cyclic multicast is initiated only after a batching delay Δ, but the multicast is terminated after at most a duration L/r, allowing the server to be idle for a new batching delay Δ that impacts only the clients whose requests arrived after the multicast began, if any. Any clients whose requests arrive during a multicast will receive part of the file during the multicast that is in progress and the rest of the file during the next multicast one batching delay Δ later, thus guaranteeing a maximum client delay of $\Delta + L/r$. A multicast is terminated before duration L/r when a client completes reception of the file and there are no remaining listeners, an event that will occur if no new client has arrived since before the previous multicast terminated. Note that the relatively simple operation of this protocol, illustrated in Figure 3.6(b), is in contrast to that of *slp*, for which the transmission schedule and service of any particular client can be extremely fragmented. The optimal value for r with *cyclic/cd,bot* is the maximum possible (b), and thus this parameter setting is used in the experiments presented.

Accurate approximations for the average server bandwidth usage and average client delay with the *cyclic/cd,bot* protocol can be derived as follows. First, two types of channel busy periods are distinguished. Channel busy periods such that at least one request arrival occurred during the preceding idle period are termed "type 1" busy periods, and will have the maximum duration L/r. The remaining busy periods are termed "type 2" busy periods. A type 2 busy period will have duration equal to L/r if there is at least one request arrival during this period. If there are no such arrivals, the duration will equal the maximum, over all clients whose requests arrived during the preceding busy period, of the amount of data that the client has yet to receive, divided by r.

Now, make the approximation that the rate at which a type 2 busy period ends when prior to its maximum duration L/r (i.e., the system empties) is constant. Denoting this rate by γ, the probability that a type 2 busy period is of duration less than *L*/*r* (also equal to the probability that the system empties during this busy period), is then given by $1-e^{-\gamma L/r}$, and the average duration of a type 2 busy period is given by $(1-e^{-\gamma L/r})/\gamma$. Note that the duration of a type 2 busy period of less than maximum duration depends only on the duration of the previous busy period and the points at which request arrivals occurred during this previous period. In light of this observation, suppose that α is independent of Δ, and calculate α for a system with $\Delta \to 0$. Consider, for $\Delta \to 0$, the average total duration of a type 1 busy period and the following type 2 busy periods up to when the system next empties (following which there is the next type 1 busy period). This quantity is equal to the average duration of an *M*/*G*/∞ busy period with arrival rate λ and service time *L*/*r*, as given by $(e^{\lambda L/r}-1)/\lambda$. This quantity is also equal to the probability that the total duration is greater than *L*/*r* (equal to $1-e^{-\lambda L/r}$) times the average total duration conditioned on being greater than *L*/*r* (equal to *L*/*r*+1/γ), plus the probability that the total duration is equal to *L*/*r* (equal to $e^{-\lambda L/r}$) times *L*/*r*, yielding

$$\left(e^{\lambda L/r}-1\right)/\lambda=\left(1-e^{-\lambda L/r}\right)\left(L/r+1/\gamma\right)+\left(e^{-\lambda L/r}\right)L/r \quad \Rightarrow \quad \gamma=\frac{1-e^{-\lambda L/r}}{\left(e^{\lambda L/r}-1\right)/\lambda-L/r}. \tag{3.20}$$

Let $p_{emptied}$ denote the probability that at the beginning of a randomly chosen idle period the system had emptied; i.e., there were no clients with unsatisfied requests. Let p_{type1} denote the probability that a randomly chosen busy period is of type 1. These two probabilities can be obtained by solving the following two equations, the first of which applies $p_{emptied}$ to the idle period preceding a randomly chosen busy period, and the second of which applies p_{type1} to the busy period preceding a randomly chosen idle period:

$$p_{type1}=p_{emptied}+\left(1-p_{emptied}\right)\left(1-e^{-\lambda\Delta}\right); \tag{3.21}$$

$$p_{emptied}=p_{type1}e^{-\lambda L/r}+\left(1-p_{type1}\right)\left(1-e^{-\gamma L/r}\right). \tag{3.22}$$

The average duration of a channel busy period is given by $p_{type1}L/r+(1-p_{type1})(1-e^{-\gamma L/r})/\gamma$ and the average duration of an idle period by $p_{emptied}/\lambda+\Delta$, yielding

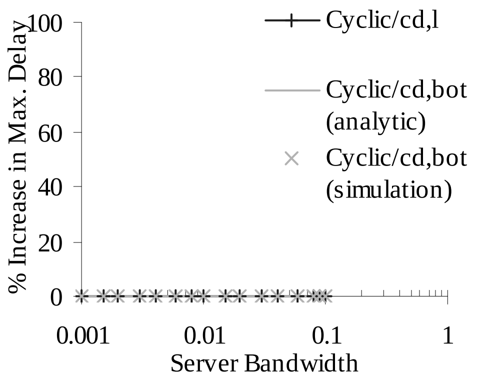
$$B_{c/cd,bot}=r\frac{p_{type1}L/r+\left(1-p_{type1}\right)\left(1-e^{-\gamma L/r}\right)/\gamma}{p_{type1}L/r+\left(1-p_{type1}\right)\left(1-e^{-\gamma L/r}\right)/\gamma+p_{emptied}/\lambda+\Delta}; \tag{3.23}$$

$$A_{c/cd,bot} = \frac{\Delta\left(p_{emptied} + \lambda\Delta/2\right) + \left(1 - p_{emptied}\right)\frac{(\lambda L/r)}{1 - e^{-\lambda L/r}}\Delta}{\lambda\left(p_{type1}L/r + (1 - p_{type1})(1 - e^{-\gamma L/r})/\gamma + p_{emptied}/\lambda + \Delta\right)} + L/r\,; \tag{3.24}$$

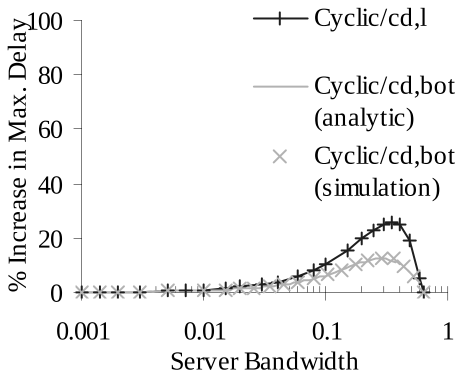
$$D_{c/cd,bot} = \Delta + L/r\ . \tag{3.25}$$

The derivation of the first term in the numerator of the equation for average delay is similar to the corresponding term in the average delay equations for *batching/cbd* and *cyclic/cd,l*, except that the batching delay was triggered by a new request arrival (which then experience the maximum waiting time Δ) only in the case when the system has emptied (with probability $p_{emptied}$). The second term in the numerator is the probability that at the beginning of a randomly chosen idle period the system had not emptied (i.e., that the idle period results from the limit of *L*/*r* on the duration of a multicast), times the average number of clients still active at the beginning of such an idle period all of whom must wait until the next multicast to complete their service, times the duration Δ of this wait. The average number of clients active at the beginning of such an idle period is equal to the average number of arrivals during the preceding busy period (of length *L*/*r*), conditioned on there being at least one such arrival.

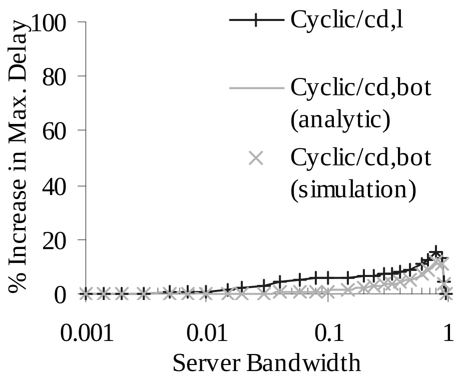
The results in Figure 3.7 show that the *cyclic/cd,bot* protocol performs close to optimal (within 15% in all cases). The figure also illustrates the high accuracy of the approximate analysis. In addition, Figure 3.7 illustrates that even the simple hybrid *cyclic/cd,l* protocol can yield good performance (within 30% of optimal in all cases), although note that the results shown for this protocol are with optimal parameter settings. An advantage of *cyclic/cd,bot* is that it has just one parameter (Δ), which is chosen based on the desired trade-off between maximum delay and bandwidth usage. Since *cyclic/cd,bot* is relatively simple and outperforms *cyclic/cd,l*, the performance of *cyclic/cd,l* with alternative (suboptimal) parameter settings is not explored here.



(a) $b = 0.1$



(b) $b = 1$



(c) $b = 10$

Figure 3.7: Maximum Delay with Improved Protocols Relative to Lower Bound (L = 1, λ = 1).

3.3.2 Protocols Minimizing Average Delay

Again, consider first a simple hybrid of batching and cyclic multicast in which a cyclic multicast is initiated only after a batching delay, in this case of the same form as in the *batching/rbd* protocol from Section 3.1.1, and terminated when there are no active clients (as in the *cyclic/l* protocol). The average server bandwidth and average/maximum client delay achieved with this *cyclic/request-based delay, listeners (cyclic/rbd,l)* protocol, with batching delay parameters n and f (integer $n \geq 1$, $0 \leq f < 1$), and transmission rate r ($r \leq b$), are given by

$$B_{c/rbd,l} = r\frac{e^{\lambda L/r}-1}{e^{\lambda L/r}-1+(n+f)}; \tag{3.26}$$

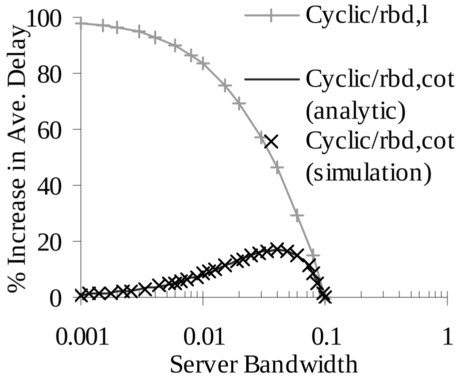
$$A_{c/rbd,l} = \frac{n(n+2f-1)/(2\lambda)}{e^{\lambda L/r}-1+(n+f)}+L/r; \qquad D_{c/rbd,l} \text{ is unbounded.} \tag{3.27}$$

These expressions are derived using the average duration of a channel busy period ($e^{\lambda L/r}-1$)/λ and the average duration of an idle period (n+f)/λ. As with the *cyclic/cd,l* protocol, r = b is not necessarily optimal, and parameter settings that optimize for average delay are found numerically.

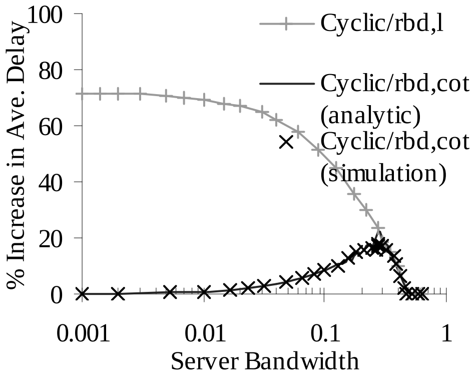
The key to designing a better protocol is, as before, determining a better policy for when to stop transmitting. If the total time each client spent receiving data from the channel was exponentially distributed (rather than of constant duration L/r), then the optimal policy for average delay would be for the server to continue its cyclic multicast whenever there is at least n (or n+1 for some fraction of busy periods f) clients with unfulfilled requests. In the (actual) case of constant service times, however, the objective of achieving consistently good sharing of multicasts has to be balanced by consideration of the required remaining service time of the active clients. For example, if a client has only a small amount of additional data that it needs to receive for its download to complete, then continuing the cyclic multicast may be optimal with respect to the average delay metric regardless of the number of other active clients.

In the protocol proposed here, termed *cyclic*/*request-based delay*, *controlled on-time* (*cyclic/rbd,cot*), these factors are roughly balanced by distinguishing between clients whose requests were made prior to the beginning of a busy period, and clients whose requests were made during it. The server continues its cyclic multicast at least until all of the former clients complete their downloads (time L/r), after which transmission continues only as long as the number of clients with unfulfilled requests is at least max[n-1, 1], where n is the same as the batching delay parameter that is used, together with the parameter f, to control the initiation of transmission after an idle period. Empirically, the optimal r is equal to b for this protocol.

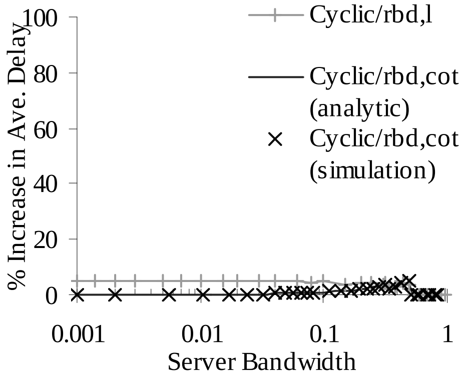
Note that for n = 1 or 2, this protocol is identical to the *cyclic/rbd,l* protocol with r = b, the analysis of which was given above. Although an exact analysis of this protocol for $n \geq 3$ appears to be quite difficult, an accurate approximate analysis has been developed. This approximate analysis constrains the duration of a busy period to be a multiple of L/b, yielding the following approximations for server bandwidth usage and average client delay (for $n \geq 3$):



(a) $b = 0.1$



(b) $b = 1$



(c) $b = 10$

Figure 3.8: Average Delay with Improved Protocols Relative to Lower Bound (L = 1, λ = 1).

$$B_{c/rbd,cot} = b\frac{(1/p)L/b}{(1/p)L/b + \left(n + f - \sum_{i=0}^{n-2} i\frac{p_i}{p}\right)\Big/\lambda}\,; \tag{3.28}$$

$$A_{c/rbd,cot} = \frac{L}{b} + \frac{\sum_{i=0}^{n-2}\frac{p_i}{p}\left[\,i(n-i)\frac{1}{\lambda} + \frac{(n-i)(n-i-1)}{2\lambda} + \frac{n}{\lambda}f\,\right]}{\lambda\left((1/p)L/b + \left(n + f - \sum_{i=0}^{n-2} i\frac{p_i}{p}\right)\Big/\lambda\right)}, \tag{3.29}$$

where $p_i = \frac{1}{i!}(\lambda L/b)^i e^{-\lambda L/b}$ and $p = \sum_{i=0}^{n-2} p_i$. $D_{c/rbd,cot}$ is unbounded. The derivations of these expressions are analogous to those for the *shifted arrivals* protocol in Section 3.2.2.

The results in Figure 3.8 show that the *cyclic/rbd,cot* protocol yields performance close to optimal, with an average delay within 20% of the lower bound in

Table 3.2: Summary of Worst-case Performance ($\partial_{f>0} = 1$ if $f > 0$ and 0 otherwise).

Protocol	Parameters	Average Client Delay	Average Server Bandwidth
Batching/cbd	Δ, r	$\Delta + L/r$	$\min[L/\Delta, \lambda L]$
Batching/rbd	n, f, r	$(n-1+\partial_{f>0})/\lambda + L/r$	$\lambda L/(n+f)$
Cyclic/l	r	L/r	$\min[r, \lambda L]$
Cyclic/cd,l	Δ, r	$\Delta + L/r$	$\min[r, \lambda L]$
Cyclic/cd,bot	Δ, r	$\Delta + L/r$	$\min[L/(\Delta+L/r), \lambda L]$
Cyclic/rbd,l	n, f, r	$(n-1+\partial_{f>0})/\lambda + L/r$	$\min[r, \lambda L]$
Cyclic/rbd,cot	n, f, r	$(n-1+\partial_{f>0})/\lambda + L/r$	$\min[r, \lambda L/\max[\mathrm{n}-1,1]]$

all cases considered. Note also that the lower bound on average delay is achievable only for high server bandwidth (low delay), specifically the region in which the *cyclic/w*2*,l* protocol operates, so performance is even closer to optimal than these results would suggest. Also shown in the figure is the high accuracy of the approximate analysis. Finally, the figure shows that the simple hybrid *cyclic/rbd,l* protocol yields good performance across the server bandwidth range of most interest only for high client reception rates (i.e., rates such that the probability of a client request arrival during the time required to download the file is very low).

3.3.3 Worst-case Performance

This section relaxes the Poisson arrival assumption and considers the worst-case performance of the protocols under arbitrary request arrival patterns. Specifically, of interest is the worst-case average server bandwidth usage and average client delay, as functions of the protocol parameters and the average request rate λ. The results are summarized in Table 3.2. The maximum client delay is not considered, since for each protocol either the maximum delay is independent of the request arrival pattern, or it is unbounded under Poisson arrivals and can therefore be no worse with some other arrival process. Note that achieving these worst-case results often requires the arrival pattern to be pessimally tuned according to the values of the protocol parameters, and that the worst-case average bandwidth usage and the worst-case average client delay cannot usually be achieved with the same arrival pattern.

Consider first the average client delay. For *cyclic*/*l*, the client delay (and thus the average client delay) is always L/r. For *batching*/*cbd*, *cyclic*/*cd,l*, and *cyclic*/*cd,bot*, the average client delay can be at most the maximum client delay, and this is achieved when all request arrivals occur in batches (of arbitrary size) with each batch arriving when there are no previous clients with outstanding requests. For *batching*/*rbd*, *cyclic*/*rbd*,*l*, and *cyclic*/*rbd*,*cot*, consider first the case of $f = 0$. With all three protocols, note that the average client delay cannot exceed $(n-1)/\lambda+L/r$, since in that case the average number of clients waiting for a multicast transmission to begin would (from Little's Law) exceed $n-1$, whereas in each protocol there can never be more than $n-1$ waiting clients. An arrival pattern for which this average client delay is achieved is as follows. Immediately after the end of a multicast transmission, a batch of $n-1$ requests arrives. Following this batch arrival a long delay ensues of deterministic duration $((n-1+m)/\lambda)-L/r$, for $m\to\infty$, followed by a batch arrival with m requests. This initiates a new multicast transmission of duration L/r. It is straightforward to verify that the average arrival rate with this request pattern is λ and that the average client delay tends to $(n-1)/\lambda+L/r$ as $m\to\infty$. For $f > 0$, the worst-case average delay depends on the precise policy by which the server determines whether to wait until n requests have accumulated, or to wait until $n+1$ requests have accumulated, prior to beginning a new multicast, rather than just the fraction f of occasions that it waits for $n+1$. Given here is the highest possible worst-case average delay over all such policies, which can be achieved, for example, by a policy that makes the choice probabilistically. By the same argument as used above for the case of $f = 0$, the average client delay cannot exceed $n/\lambda+L/r$. An arrival pattern for which this average client delay is achieved is similar to that used above, but with a batch size of n rather than $n-1$, and (whenever the server chooses to wait for $n+1$ arrivals and thus a new transmission does not start immediately) a delay of duration $((n+fm)/\lambda-L/r)/f$, for $m\to\infty$, followed by a batch arrival with m requests.

Consider now the average server bandwidth. For *batching*/*rbd*, the average bandwidth depends only on the average arrival rate, rather than the specific arrival pattern, since every n^{th} (or $n+1^{st}$) request arrival causes a new transmission of the file that only the clients making those n (or $n+1$) requests receive. Thus, the worst-case average bandwidth usage for this protocol is the same as the average bandwidth usage for Poisson arrivals. For *batching*/*cbd*, if $\lambda \leq 1/\Delta$ then request arrivals can be spaced such that no arrivals occur simultaneously and no arrivals occur during a batching delay, yielding a worst-case bandwidth usage equal to the unicast bandwidth usage of λL. For $\lambda \geq 1/\Delta$, batched arrivals with deterministic spacing of Δ between the batches yield the worst-case bandwidth usage of L/Δ. Thus, the worst-case bandwidth usage is min[L/Δ, λL]. For *cyclic*/*l*, if $\lambda \leq r/L$ the worst-case bandwidth usage is achieved when the spacing between consecutive arrivals is always at least L/r, yielding a bandwidth usage of λL. For $\lambda \geq r/L$, transmission can be continuous, giving a bandwidth usage of r. Thus, the worst-case bandwidth usage is min[r, λL]. The same worst-case bandwidth usage is achieved with *cyclic/cd,l*, and *cyclic*/*rbd,l*. For $\lambda \geq r/L$, transmission can be continuous, and for $\lambda \leq r/L$ a bandwidth usage of λL is achieved when the fraction of arrivals that occur during busy periods approaches one, and the spacing between consecutive busy-period arrivals is of deterministic duration infinitesimally less than L/r. Similarly, for *cyclic*/*rbd,cot*, if $\lambda \leq (r/L)(\max[n-1, 1])$ the worst-case bandwidth usage is achieved when the fraction of arrivals that occur during busy periods approaches one, and busy period arrivals occur in batches of size max[n–1, 1] with spacing between consecutive batches of deterministic duration infinitesimally less than L/r, yielding a bandwidth usage of $\lambda L/\max[n-1, 1]$. For $\lambda \geq (r/L)(\max[n-1, 1])$, arrivals can be spaced such that transmission is continuous, giving a bandwidth usage of r. Thus, the worst-case bandwidth usage is min[r, $\lambda L/\max[n-1, 1]$]. Finally, for *cyclic*/*cd,bot*, if $\lambda \leq 1/(\Delta+L/r)$ then request arrivals can be spaced such that no arrivals occur simultaneously or during a batching delay or channel busy period, yielding a worst-case bandwidth usage of λL. For $\lambda \geq 1/(\Delta+L/r)$, arrivals can be spaced such that the system never empties, giving a bandwidth usage of $L/(\Delta+L/r)$. Thus, the worst-case bandwidth usage is min[$L/(\Delta+L/r)$, λL].

3.4 Heterogeneous Clients

This section relax the homogeneity assumption and consider the case in which there are multiple classes of clients with differing associated maximum delays (Section 3.4.1) and achievable reception rates (Section 3.4.2). Section 3.4.1 also supposes that the amount of data a client needs to receive from a channel may be class-specific. This scenario is relevant to the protocols developed in Section 3.4.2, in which file data blocks are delivered on multiple channels and each client listens to the subset of channels appropriate to its achievable reception rate. Throughout this section only maximum client delay is considered, although the results can also yield insight for the case in which average client delay is the metric of most interest.

3.4.1 Class-specific Service Requirement and Maximum Delay

Here it is assumed that clients of class *i* have maximum delay D_i and need to receive an amount of file data L_i from a single shared channel. All clients have a common reception rate constraint *b*. As in the case of homogeneous clients, the *slp* protocol is optimal and thus its average bandwidth usage provides a lower bound on that achievable with any protocol. Section 3.4.1.1 generalizes the approximation for this lower bound that was given in Section 3.2.2, to this heterogeneous context. As motivated again by the complexity of *slp*, Section 3.4.1.2 extends the simpler and near-optimal *cyclic/cd,bot* protocol given in Section 3.3.1, so as to accommodate heterogeneous clients, and compares its performance to that of *slp*.

3.4.1.1 Lower Bound (slp) Bandwidth Approximation

A key observation used to generalize the lower bound approximation is that with *slp*, the presence or absence of requests from “high slack” clients (i.e., clients of classes *j* such that D_j is large relative to L_j/b), will have relatively little impact on the server bandwidth usage during periods with one or more active “low slack” clients. Exploiting this observation, the classes are ordered in non-increasing order of L_i/D_i, and the average server bandwidth usage of *slp*, with the assumed client heterogeneity, is written as

$$B_{slp} = \sum_{i=1}^{N_C} (P_i - P_{i-1})\,\beta_i\,, \tag{3.30}$$

where N_C denotes the number of customer classes, P_i denotes the (cumulative) probability that there is at least one client from classes 1 through *i* with an outstanding request (with P_0 defined as 0), and β_i denotes the average server bandwidth usage over those periods of time during which there is at least one client from class *i* with an outstanding request but none from any class indexed lower than *i*.

An approximation for the probability P_i can be obtained using a similar approach as was used for the corresponding quantity in the approximation for homogeneous clients. P_i is equal to the average duration of a period during which there is at least one client from classes 1 through i with an outstanding request, divided by the sum of this average duration and the average request inter-arrival time for this set of classes ($1/\sum_{k=1}^{i}\lambda_k$, where λ_k denotes the rate of requests from class k clients). The average duration of a period during which there is at least one client from classes 1 through i with an outstanding request is approximated by the average duration of an $M/G/\infty$ busy period with arrival rate $\sum_{k=1}^{i}\lambda_k$ and average service time $\sum_{j=1}^{i}\frac{\lambda_j(L_j/b)}{\sum_{k=1}^{i}\lambda_k}$, as given by ($e^{\sum_{j=1}^{i}\lambda_j L_j/b}-1)/\sum_{k=1}^{i}\lambda_k$, plus the average duration of the delay after the arrival of a request to an idle system, until the server must begin transmitting ($\sum_{j=1}^{i}\frac{\lambda_j(D_j-L_j/b)}{\sum_{k=1}^{i}\lambda_k}$), yielding

$$P_i \approx \frac{e^{\sum_{j=1}^{i}\lambda_j L_j/b}-1+\sum_{j=1}^{i}\lambda_j\left(D_j-L_j/b\right)}{e^{\sum_{j=1}^{i}\lambda_j L_j/b}+\sum_{j=1}^{i}\lambda_j\left(D_j-L_j/b\right)}. \tag{3.31}$$

The average bandwidth usage β_i is approximated as L_i reduced by the average amount of data x_i received by a class i client while there is at least one active client from a lower indexed class, divided by the portion of the time D_i during which no such lower indexed client is active:

$$\beta_i \approx \frac{L_i - x_i}{D_i\left(1-P_{i-1}\right)}. \tag{3.32}$$

Defining β_ave_i by

$$\beta\_ave_i = \sum_{j=1}^{i-1}\beta_j\left(P_j - P_{j-1}\right)/P_{i-1}, \tag{3.33}$$

the quantity x_i is computed using

$$x_i \approx \beta_i\left(D_i P_{i-1}\right)+\left(\beta\_ave_i-\beta_i\right)E_i, \tag{3.34}$$

where E_i denotes the average portion of the time D_i during which a class i client receives data from the channel at the higher average rate equal to β_ave_i, owing to the presence of requests from lower indexed classes, rather than at the lower rate β_i. A simple approximation for E_i would be D_iP_{i-1}, but this would neglect the impact of variability in the portion of time t_i that there is at least one active client from a lower indexed class, during the period over which a particular class i client is active. In particular, there is a maximum length of time during which a class i client can receive data at the higher average rate, without accumulating an amount of data exceeding L_i. Noting that t_i is at most D_i, the first-order impact of variability is captured by assuming a truncated exponential distribution for t_i, with rate parameter φ_i such that the average of the distribution is $D_iP_{i\text{-}1}$:

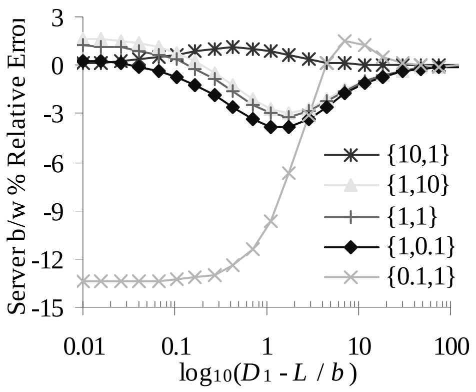
$$\frac{1}{\varphi_i} - \left(D_i e^{-\varphi_i D_i}\right) \Big/ \left(1 - e^{-\varphi_i D_i}\right) = D_i P_{i-1}\,. \tag{3.35}$$

During the portion of time when a class i client is receiving data at the higher average rate β_ave_i, the rate at which additional data is received (i.e., beyond that which would otherwise be received) is given by $\beta\_ave_i – \beta_i$. Since at most L_i data can be received in total during this time period, the average additional amount of data that can be received owing to reception at the higher average rate is upper bounded by $L_i – E_i\beta_i$. Here the maximum length of time during which a class i client can receive data at the higher average rate, without accumulating an amount of data exceeding L_i, is approximated by t_max_i = min[D_i, $(L_i – E_i\beta_i)/(\beta\_ave_i – \beta_i)$]. An approximation for E_i is then obtained as

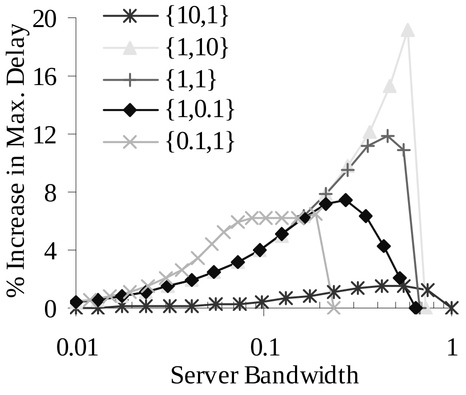
$$E_i \approx \left(\frac{\left(1 - e^{-\varphi_i t\_max_i}\right) / \varphi_i - t\_max_i \left(e^{-\varphi_i D_i}\right)}{1 - e^{-\varphi_i D_i}} \right) + t\_max_i \left(\frac{e^{-\varphi_i t\_max_i} - e^{-\varphi_i D_i}}{1 - e^{-\varphi_i D_i}} \right), \tag{3.36}$$

where the first term is the probability that t_i does not exceed t_max_i times its expected value in this case, and the second term is t_max_i times the probability that t_i exceeds t_max_i.

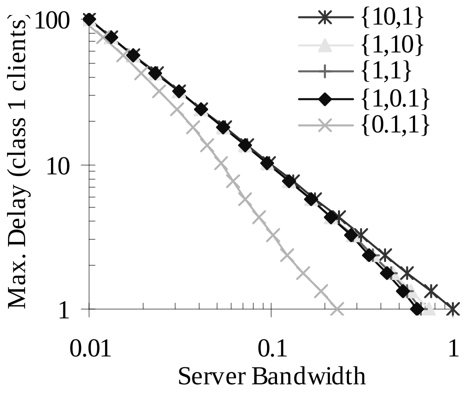
The above analysis results in a system of non-linear equations that can easily be solved numerically, beginning with the quantities for class 1 and proceeding to those for successively higher indexed classes. Although the analysis might seem complex, simpler variants were found to have substantially poorer accuracy. Note also that for the case in which the client classes have identical L_i and D_i, the analysis yields identical bandwidths β_i, and the bound reduces to that given earlier for homogeneous clients.



(a) Accuracy of Lower Bound Approximation



(b) Maximum Delay with *Cyclic/cd,bot* Relative to Lower Bound



(c) Relative Impact of Low Slack vs. High Slack Clients

Figure 3.9: Impact of Class-specific Maximum Delays (L = 1, D_2 = $5D_1$, varying arrival rates $\{\lambda_1, \lambda_2\}$).

Sample validation results comparing the analysis against simulations of the *slp* protocol are presented in Figure 3.9(a). In the scenarios considered in this figure there are two client classes, with $L_1 = L_2 = 1$ and $D_1 = 5D_2$. The maximum sustainable client reception rate b is fixed at one. Five combinations of request rates $\{\lambda_1, \lambda_2\}$ are considered, and the percent relative error in the average server bandwidth usage computed using the approximate analysis is plotted against the slack (D–L/b) of the low slack clients (class 1), for each request rate combination. Additional experiments included a full factorial experiment for two class systems, and an experiment in which a large number of randomly generated systems with 3-6 classes were tested. In all these experiments no case was found in which the absolute relative error exceeded 20%.

3.4.1.2 Extension of Cyclic/cd,bot

The *cyclic/cd,bot* protocol is extended to accommodate heterogeneous clients as follows. The duration of each multicast transmission is limited to at most the maximum value of L_i/r over all classes i that have active clients at the beginning of the transmission. As before, if the last active client completes reception of the file and there are no more listeners, the transmission is terminated early. The delay Δ becomes variable, now being dependent on which classes have clients with outstanding requests. At the beginning of each delay period, it is initialized to the minimum value of $D_i – L_i/r$ over all classes i that have active clients. If a client of some other class j arrives during the delay period, and the time remaining in the delay period exceeds $D_j – L_j/r$, the length of the delay period must be reduced accordingly. As before, each client obtains the entire file either in a single busy period, or in two busy periods separated by an idle period, and the optimal r is equal to b.

Representative simulation results comparing performance with the extended *cyclic/cd,bot* protocol to the lower bound defined by the optimal *slp* protocol are presented in Figure 3.9(b). (The analytic approximation from Section 3.4.1.1 is not used here, as the differences from optimality of *cyclic/cd,bot* are not sufficiently greater than the errors in the approximation.) As in Figure 3.9(a), there are two client classes with $L_1 = L_2 = 1$ and $D_1 = 5D_2$, the client reception rate b is fixed at one, and five combinations of request rates $\{\lambda_1, \lambda_2\}$ are considered. As the figure illustrates, the achieved performance is reasonably close to optimal.

Figure 3.9(c) shows the maximum delay for class 1 clients (the maximum delay for class 2 clients is five times greater) as a function of server bandwidth for the *cyclic/cd,bot* protocol, for the same scenarios as previously considered. Noting that the curves can be separated into three groups based only on the request rate of the low slack clients, the main observation from this figure is the minimal impact of the request rate of the “high slack” clients on system performance.

3.4.2 Class-specific Reception Rates

Suppose now that class i clients have a class-specific maximum sustainable client reception rate b_i as well as maximum delay D_i, but common $L_i = L$. Section 3.4.2.1 presents an algorithm for computing a lower bound on the required average server bandwidth. In Section 3.4.2.2, scalable download protocols for this context are proposed and their performance evaluated.

3.4.2.1 Heterogeneous Lower Bound

The *slp* protocol can be suboptimal when there is heterogeneity in client reception rates. For example, consider a scenario in which two clients request the file at approximately the same time, one with a relatively high reception rate and a relatively low maximum delay and one with a low reception rate and a high maximum delay, and in which no other requests arrive until these two clients have completed reception. With *slp*, the server will delay beginning transmission for as long as possible, and then, if it is the high-rate client that has no slack at this point, begin transmitting at an aggregate rate equal to the rate of the high-rate client. However, in this case greater sharing of the server transmissions, and thus lower server bandwidth usage, could be achieved by starting transmission earlier, at the low rate.

Using the notation in Table 3.3, Figure 3.10 presents an algorithm that yields a lower bound on the server bandwidth required to serve a given sequence of request arrivals[16]. The algorithm considers each request j in order of request deadline; i.e., the time by which the associated client must have completed reception of the file so as not to exceed the maximum delay for its respective class. The quantity x_j^{hlb} approximates (in a manner allowing a lower bound on server bandwidth to be computed) the amount of additional data (not received by earlier clients) that the server would need to transmit to enable the request j client to meet its deadline. This quantity is computed as $L - y_{j-1,j}^{hlb}$, where $y_{j-1,j}^{hlb}$ is the total over all earlier requests k of an optimistic estimate $x_{k,j}^{hlb}$ of the portion of x_k^{hlb} that the request j client could have shared reception of. A proof that B_j^{hlb}

[16] The algorithm as presented in Figure 10 has complexity O(K^2), but can easily be implemented in a more efficient manner in which only requests i whose time in system overlaps with that of request j are explicitly considered in the inner loop.

Table 3.3: Notation for Heterogeneous Lower Bound Algorithm.

Symbol	**Definition**
K	Length of request sequence, with requests indexed from 1 to K in order of request deadline
$c(j)$	The class of the request j client
T_j^A	Arrival time of request j
T_j^D	Deadline of request j ($T_j^A + D_{c(j)}$)
$T_{j,i}$	Time from the arrival of request i until the deadline of request j ($T_j^D - T_i^A$)
x_j	Amount of data received by the request j client, from transmissions not received by any client with an earlier request deadline
$x_{j,i}$	Amount of data received by the request j client, from transmissions not received by any client with an earlier request deadline, that is also received by the request i client ($j < i \le K$)
$y_{j,i}$	Sum of $x_{k,i}$ for $1 \le k \le j$
B_j	Total amount of data transmitted to serve requests 1 through j

```
$B_0^{hlb} = 0$, $y_{0,i}^{hlb} = 0 \quad 1 \le i \le K$
for $j$=1 to $K$
  $x_j^{hlb} = L - y_{j-1,j}^{hlb}$
  $B_j^{hlb} = B_{j-1}^{hlb} + x_j^{hlb}$
  for $i$=$j$+1 to $K$
    if $T_i^A < T_j^D$ then
      $x_{j,i}^{hlb} = \min\{x_j^{hlb}, L - y_{j-1,i}^{hlb}, b_{c(i)}T_{j,i} - y_{j-1,i}^{hlb}, b_{c(i)}T_{j,j}\}$
    else
      $x_{j,i}^{hlb} = 0$
    $y_{j,i}^{hlb} = y_{j-1,i}^{hlb} + x_{j,i}^{hlb}$
  end for
end for
```

Figure 3.10: Heterogeneous Lower Bound Algorithm.

= $\sum_{k=1}^{j} x_k^{hlb}$ is a lower bound on the total server bandwidth required to serve requests 1 through j is given in Appendix A. In the case that all classes share a common maximum sustainable client reception rate, the lower bound is tight and gives the bandwidth used by *slp*. With heterogeneous client reception rates, the bound may be unachievable.

3.4.2.2 Protocols

Perhaps the simplest protocol for serving clients with heterogeneous reception rates is to dedicate a separate channel to each class. Any of the scalable download protocols from Section 3.3 can be utilized on each channel, with transmission rate chosen not to exceed the maximum sustainable reception rate of the respective clients. The disadvantage of this *separate channels* protocol is that there is no sharing of server transmissions among clients of different classes.

A second approach, termed here *shared cyclic/listeners (s-cyclic/l)*, extends the *cyclic/l* protocol from Section 3.1.2 to this heterogeneous client context. The client classes are indexed in decreasing order of their associated maximum delays, aggregating any classes with equal maximum delays into a single class. A channel is created for each class, with the transmission rate on channel 1 chosen as L/D_1 and the rate on channel i for $i > 1$ chosen as $L/D_i–L/D_{i–1}$. Class i clients listen to channels 1 through i.[17] The server cyclically multicasts file data on each channel, whenever at least one client is listening. Here (as well as for the remaining protocols discussed in this section) it is assumed that through careful selection of the order in which data blocks are transmitted on each channel [26, 27], and/or use of erasure codes with long "stretch factors", a client listening to multiple channels will nonetheless never receive the same data twice. The average server bandwidth usage on each channel i may be derived in a similar fashion as for the *cyclic/l* protocol, yielding

$$B_{sc/l} = (L/D_1)\left(1-e^{-\sum_{j=1}^{N_C}\lambda_j D_j}\right)+\sum_{i=2}^{N_C}(L/D_i - L/D_{i-1})\left(1-e^{-\sum_{j=i}^{N_C}\lambda_j D_j}\right). \qquad (3.37)$$

This protocol achieves sharing of server transmissions among clients of different classes, but as with the *cyclic/l* protocol there will be periods over which transmissions on a channel serve relatively few clients.

The near-optimal protocols for delivery to homogeneous clients that were proposed in Section 3.3 have the characteristic that whenever the server transmits, it is at the maximum client reception rate *b*. Intuitively, for fixed maximum or average client delay, transmitting at the maximum rate allows a greater delay before beginning any particular transmission, and thus a greater opportunity for batching. In contrast, note that in the *s-cyclic/l* protocol, clients of each class i receive server transmissions that are at an aggregate rate equal to the *minimum* rate required to complete their downloads within time D_i. The key to devising an improved protocol is to achieve a

[17] Alternatively, a large number of channels may be employed, with the server transmitting on each at the same low rate r. Class i clients would then listen to channels 1 through k_i, where $k_i = \lceil L/(rD_i)\rceil$.

good compromise between use of higher aggregate rates, which permit better batching opportunities for the clients that can receive at those rates, and low aggregate rates that maximize the sharing of server transmissions among clients of different classes.

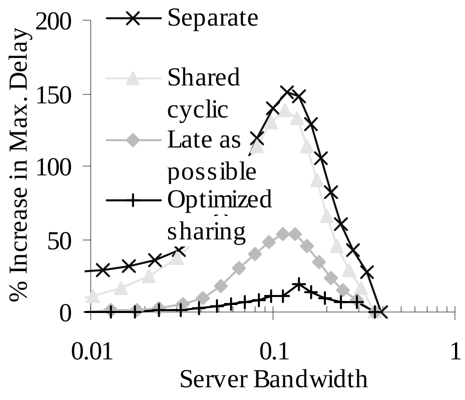
A family of protocols that enables such a compromise is defined as follows. The client classes are indexed in non-decreasing order of their reception rates. A channel is created for each client class, with the transmission rate r_i on channel i chosen as $b_i - \sum_{j=1}^{i-1} r_j$.[18] Class i clients receive an amount of data l_i^j on each channel j, for $1 \le j \le i$, as determined by the protocol, such that $L = \sum_{j=1}^{i} l_i^j$. Server transmissions on each channel follow a protocol such as the extended *cyclic/cd,bot* protocol from Section 3.4.1.

Within this family, two extremes can be identified. At one extreme, clients receive the maximum amount of data possible on the lower-numbered channels, thus maximizing the sharing of transmissions among clients of different classes. Specifically, class i clients receive an amount of data $l_i^j = \min[L - \sum_{k=1}^{j-1} l_i^k, r_j D_i]$ on each channel j, $1 \le j \le i$.[19] At the other extreme, batching opportunities for class i clients are maximized by equalizing their slack on each channel. In this case, $l_i^j = (r_j/b_i)L$ for each channel j, $1 \le j \le i$. Simulation results have shown that neither of these protocols yields uniformly better performance than the other, and that the performance differences between them can be quite significant.

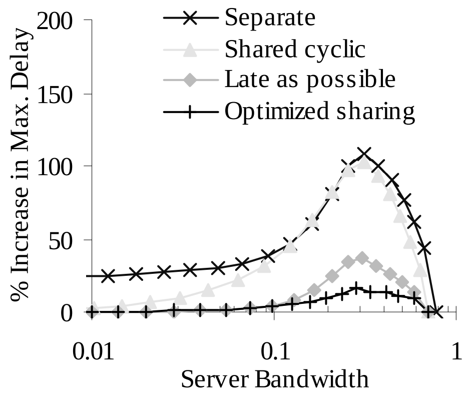
The best intermediate strategy can be closely approximated by a protocol termed here *optimized sharing*, in which the l_i^j values are chosen to be approximately optimal. With N_C classes, the number of free parameters in the optimization problem is $N_C(N_C-1)/2$. For each candidate allocation, the approximate lower bound analysis from Section 3.4.1 can be used to estimate the average server bandwidth with that allocation. With a small number of classes, as in the experiments whose results are presented here, L can be discretized and exhaustive search employed, for example, to find an allocation that results in the minimum predicted average server bandwidth.

[18] Note that if $b_i = b_{i-1}$, then the rate r_i is computed as 0. Channel i will then not be used, but for convenience of indexing it is retained.

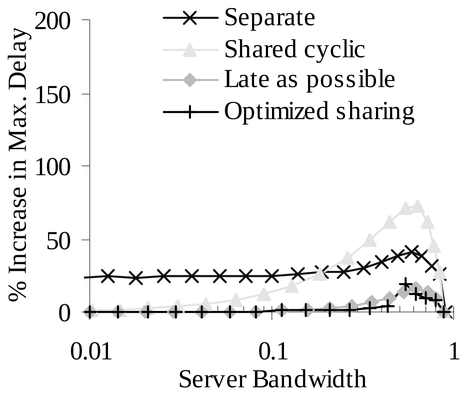
[19] If this rule results in class i clients retrieving no data from channel i, then channel i can effectively be aggregated with channel i+1.



(a) 80% b=0.2; 10% b=1; 10% b=5



(b) equal split among b = 0.2, 1, 5



(c) 10% b = 0.2; 10% b=1; 80%

Figure 3.11: Maximum Delay with Heterogeneous Client Protocols Relative to Lower Bound (L = 1, λ = 1, D values inversely proportional to maximum achievable reception rates).

Note that with all of the above protocols, the amount of data received on each channel by a client is statically determined according to the client's class. The extension to heterogeneous clients of the *slp* protocol, in which a client's use of each channel is dynamically determined, is also considered. The client classes are indexed in non-decreasing order of their associated maximum sustainable reception rates. The server transmits at aggregate rate b_i whenever there is at least one client from class i that has no slack, and there is no such client from a class indexed higher than i. Channels are defined (as in the previous protocol family, for example), such that a class j client can receive at rate min[b_i, b_j] whenever the server is transmitting at aggregate rate b_i.

Figure 3.11 shows representative performance results, using the heterogeneous lower bound algorithm from Section 3.4.2.1 to provide a baseline for comparison. For the *separate channels* and *optimized sharing* protocols, the optimal *slp* protocol is used on each channel, although as illustrated in Figure 3.9(b) use of the more practical *cyclic/cd,bot* protocol would not greatly impact the results. For the heterogeneous client *slp* protocol and for *optimized sharing*, simulation is used to obtain the results shown (although as noted previously, the approximate lower bound analysis is used in *optimized sharing* to determine the data allocation), while for *separate channels* and *s-cyclic/l*, the results are from analysis. In the scenarios considered in this figure there are 3 client classes with respective reception rates of 0.2, 1, and 5, and D values such that $b_iD_i = b_jD_j$ for all classes i, j. The total request arrival rate is (without loss of generality) fixed at one, and the different parts of the figure correspond to different choices for the division of the total request rate among the classes.

The principal observations from this figure are: (1) the *separate channels* protocol yields poor performance, even in this scenario with greatly differing client reception rates; (2) the *s-cyclic/l* protocol can yield performance as poor, or worse than, *separate channels* (note, however, that the protocol does relatively better when the classes are more similar); (3) the *optimized sharing* protocol yields substantially better performance than *separate channels* and *s-cyclic/l*, and never worse (and sometimes significantly better) than the heterogeneous client *slp* protocol; and (4) the *optimized sharing* protocol does not appear to leave much room for performance improvement, achieving within 25% of the lower bound on maximum client delay in all scenarios considered.

3.5 Summary

This chapter considers the problem of using scalable multicast protocols to support on-demand download of large files from a single server to potentially large numbers of clients. Lower bounds are developed that indicate the best achievable performance. Baseline batching and cyclic multicast protocols are found to have significantly sub-optimal performance, motivating the development of new protocols. In the case of homogeneous clients, the best of the new practical protocols that focus on improving maximum client delay yields results within 15% of optimal, in all scenarios considered. Similarly, the best of the new protocols designed to improve average client delay yields results within 20% of optimal. For heterogeneous clients, the proposed *optimized sharing* protocol achieves within 25% of the optimal maximum client delay, in all scenarios considered. An interesting observation is that it can substantially outperform the *slp* protocol, which is optimal in the homogenous environment.

Chapter 4
Scalable Download from Multiple Replicas

Systems using replication require a *replica selection* policy that chooses a replica to serve each client request. In systems using aggregation as well as replication, a basic tradeoff that the replica selection policy must address is between locality of service and efficiency of service. At one extreme, each client request could be served by the closest replica, maximizing locality of service. At the other extreme, all client requests could be served by the same replica, maximizing opportunities for aggregation and thus efficiency of use of server resources. In intermediate policies, some client requests are served by the closest replica, and others are served by replicas at which a higher degree of aggregation can be achieved.

This chapter considers the problem of replica selection in systems utilizing both replication and aggregation. As discussed in Section 2.3.2, prior work on the replica selection problem has assumed individual rather than aggregated service [35, 93, 94, 96, 133, 140, 186], or has considered aggregated service but only in the specific context of media streaming and corresponding streaming-based service aggregation techniques [9, 72, 81]. In contrast to assuming a media streaming context, this chapter considers two general types of service aggregation that may be applicable in a variety of contexts, and in particular to systems providing a *download* service for large files, such as software distributions or videos. In the case of download, the two service aggregation types considered correspond to: (a) batching multiple requests for the same file and serving them with a single (IP or application-level) multicast, or (b) using a "digital fountain" approach [31, 146, 176], respectively. For each service aggregation type, classes of policies of differing complexities are compared, with the goal of determining the performance improvements that more complex types of policies may enable. Comparisons are carried out in the context of a simple system model that allows the performance that may be achievable with each class of policies to be accurately delimited.

The first and most basic policy distinction considered is between replica selection policies that use *dynamic* state information (for example, numbers of waiting requests), and (simpler) policies that use only *static* (or semi-static) client-replica proximity and average load information. The obtained results indicate that use of dynamic state information has the potential to yield large reductions in client delay, for fixed total service delivery cost, in many cases by a factor of two or more.

Among policies using dynamic state information, a second distinction can be made between policies that *defer* replica selection decisions, for example until near or at the end of a batching delay as employed by the aggregation policy, and those (simpler) policies that make a replica selection decision immediately upon request arrival. It is found that deferred selection can potentially yield substantial performance improvements, again by a factor of two or more in some cases, although only for fairly narrow ranges of model parameter values.

Finally, among policies using dynamic state information and deferred selection, a third distinction is between "local state" policies that base their replica selection and scheduling decisions on the currently outstanding "local" client requests, and "global state" policies that use information concerning all current requests. It is found that relatively simple local state policies appear able to achieve most of the potential performance gains.

The remainder of the chapter is organized as follows. The system model is described in Section 4.1. Section 4.2 addresses the question of whether use of dynamic state information can yield major performance improvements. Section 4.3 considers the extent to which performance can potentially be improved in dynamic policies by deferring replica selection decisions, rather than making such decisions immediately upon request arrival. Section 4.4 focuses on the class of policies using dynamic state information and deferred selection, and considers the extent to which polices using only "local" state information can realize the full potential of this policy class. Throughout Sections 4.2, 4.3, and 4.4, the maximum client delay is the primary metric used to measure client performance. Section 4.5 considers the impact of using average delay as delay metric. A short summary is presented in Section 4.6.

4.1 System Model

Consider a system with N replicas, each of which delivers a service (such as download of a popular file) using the same service aggregation technique. Clients are divided according to network location into M groups, such that all of the clients in a group can be considered to have approximately the same network proximity to each replica. For simplicity, in the following it is assumed that $M = N$. Given this assumption, the client groups and replicas are indexed such that for the clients of group i, the closest replica is replica i. Replica i is called the “local” replica for client group i, while all other replicas are called “remote” replicas for this group. Service requests from the clients of each group i are assumed to be Poisson at rate λ_i, with the replicas/groups indexed from 1 to N in non-increasing order of the group request rates.

Two types of service aggregation are considered. With the first type, called *batched* service, requests are accumulated and served in batches, with each batch being served by a single replica. The required “service cost” for a batch of requests (measured in units such as processor-seconds or bytes of replica bandwidth consumed, depending on the service) is assumed to be a fixed value L, independent of the number of requests in the batch. Any request arriving after a batch has already begun service cannot receive service with that batch, but must wait for service with some other batch. For a service providing downloads of a popular file, this type of service aggregation correspond to a replica serving a batch of requests for a file of size L with a single multicast transmission.

With the second type of aggregation, called here *fountain* service, whenever a replica has at least one client wishing to receive its service, service is dispensed at a rate r to all such clients. Clients may switch replicas during their service period. As in Chapter 3, the service period is L/r. For a service providing downloads of a popular file, this type of service aggregation corresponds to using a “digital fountain” approach [31, 146, 176], in which file data is erasure encoded and transmitted by each replica at rate b on its own multicast channel whenever at least one client is listening to that channel. A requesting client need only listen to one or some sequence of replica channels for a total duration L/r, assuming use of an erasure-coding (e.g., [164]) and/or transmission scheme such that the probability of receiving duplicate packets, even when a client switches replicas during its service, is negligible.

The performance metrics considered are the maximum client delay D and the total service delivery cost. For the batched service type, the client delay is defined to include only the time from request generation until the request's batch enters service. For the fountain service type, the client delay is defined as the service duration, equal to L/r. In this case, unlike for batched service, the client delay is the same for all requests. For both service types, the total service delivery cost C is defined as the average total rate at which service cost is incurred at the replicas (in units of cost per unit time) plus the average total rate at which *access cost* is incurred. The access cost is defined in a manner that may make it applicable to a variety of service types. When a client from group i receives a fraction q of its service from a replica j (note that for batched service, q is 1 for the replica at which the client's batch is served, and 0 for all other replicas) an access cost of $c_{ij}qL$ is assumed to be incurred, where the constant c_{ij} gives the network cost per unit of service received when replica j provides service to a client from group i. For simplicity, in the following it is assumed, unless stated otherwise, that $c_{ii} = 0$, and $c_{ij} = c$ for some c such that $0 < c \leq 1$, for all $i \neq j$. B_i is used to denote the average rate at which service cost is incurred at each replica i. Considering download systems, as in Chapter 3, B_i corresponds to the average server bandwidth usage at a replica i. Using the notation defined in Table 4.1, this yields a total service delivery cost C calculated as

$$C = \sum_{i=1}^{N} B_i + \left(\sum_{i=1}^{N} \lambda_i q_i \right) cL . \tag{4.1}$$

Clearly, there is a tradeoff between maximum client delay and total service delivery cost, and in policy comparisons either the maximum client delays can be compared, for a fixed total service delivery cost, or the total service delivery costs can be compared, for a fixed maximum client delay.

Table 4.1: Notation used in Chapter 4

Symbol	**Definition**
λ_i	Request rate from the clients of group *i*; groups indexed so that $\lambda_i \geq \lambda_j$ for $i \leq j$
λ	Total request rate, summed over all client groups
L	Service required by each requesting client (equal to the file size)
N	Number of replicas (assumed equal to the number of client groups)
B_i	Average rate at which service cost is incurred at replica *i* (assumed equal to the average server bandwidth)
c	Access cost per unit of service received for all client groups *i* and replicas *j*, $i \neq j$
q_i	Average fraction of its service that a group *i* client receives from other than replica *i*
C	Total service delivery cost
D	Maximum client delay
A	Average client delay
r	Service rate with fountain service

4.2 Dynamic vs. Static Policies

Static policies use only client-replica proximity and average load information in making replica selection decisions. Although fountain service, in general, allows each client to switch replicas during its service period, with static policies there can be no advantage to this flexibility, and therefore with such policies it is assumed that each request is served by a single replica, for both batched and fountain service. Furthermore, in a static policy either all requests from a given client group are served by the same replica (in general, dependent on the group), or replica selection is probabilistic. In either case, with Poisson requests from each client group, request arrivals at each replica will also be Poisson. Section 4.2.1 reviews the prior analysis results for a single replica with Poisson request arrivals. In Section 4.2.2, these results are applied to determine a tight bound on the achievable performance with static policies for each of the batched and fountain aggregation types. Section 4.2.3 accurately delimits the achievable performance with dynamic policies. Performance comparisons are presented in Section 4.2.4.

4.2.1 Analysis for Single Replica Systems

Referring to the single server analysis for the batched and cyclic multicast (for the “fountain service” case) presented in Chapter 3, the service cost (e.g., average server bandwidth usage) can easily be calculated.

Consider first the case of batched service. For a fixed maximum client delay (waiting time) *D*, the delivery cost is minimized by a policy in which all currently waiting requests are served once the waiting time of the earliest such request reaches *D*. Since the expected time duration from the beginning of service of one batch until the beginning of service of the next is $D + 1/\lambda$ with this policy, the following equation relates the optimal *D* and average service cost rate *B* (equation (3.1) and (3.2)):

$$B = \frac{L}{D + 1/\lambda}. \tag{4.2}$$

For fountain service, the replica is dispensing service at a rate *r* whenever there is at least one client with a request that is not yet satisfied. The choice of *r* determines the achieved tradeoff between the client delay and the average service cost rate. Since each request has a required service time of *L*/*r*, and the probability of there being no active request (and thus of the replica being idle) is $e^{-\lambda L/r}$, the following equations are obtained for *D* and *B* (equation (3.5) and (3.6)):

$$D = L/r; \qquad\qquad B = b\left(1 - e^{-\lambda L/r}\right). \tag{4.3}$$

4.2.2 Delimiting the Achievable Performance with Static Policies

Note that in equations 4.2 and 4.3, the average service cost rate *B* given fixed *D* is a monotonically increasing, concave function of the request arrival rate at the replica. Thus, in a static policy, if all requests that are served by a remote replica are served by the replica with the highest rate of requests from its local client group (i.e., replica 1), the total of the average service cost rates at the replicas will be minimized. Furthermore, since assuming that $c_{ij} = c$ for all $i \neq j$, serving such requests at replica 1 incurs no greater access cost than serving them at any other remote replica(s). Finally, the concavity of the average service cost rate function, and the assumptions regarding access costs, imply that in an optimal static policy either all requests from a client group are served by a remote replica (namely, replica 1), or none are, and the former case can hold only if all requests from client groups with equal or lower request rate are also served remotely. Thus, in an optimal static policy there is an index k ($1 \le k \le N$), such that all requests from group i clients, for $i \le k$, are served by the local replica, while all requests from group j clients, for $j > k$, are served by replica 1. Note that for homogenous systems in which the client groups have identical request rates, in the optimal static policy either all requests are served by the local replica, or all requests are served at some single replica.

Given the form of the optimal static policy as described above, from equations (1) and (2) a tight lower bound on the total service delivery cost achievable with a static policy and batched service, for a fixed maximum client delay D, is given by

$$\min_{k=1,2,\cdots,N}\left\{\frac{L}{D+1/\left(\lambda_1+\sum_{i=k+1}^{N}\lambda_i\right)}+\sum_{i=2}^{k}\frac{L}{D+1/\lambda_i}+c\sum_{i=k+1}^{N}\lambda_i L\right\}. \tag{4.4}$$

Equations (1) and (3) yield the corresponding expression for fountain service, where $D = L/r$:

$$\min_{k=1,2,\cdots,N}\left\{b\left(1-e^{-\left(\lambda_1+\sum_{i=k+1}^{N}\lambda_i\right)L/r}\right)+\sum_{i=2}^{k}b\left(1-e^{-\lambda_i L/r}\right)+c\sum_{i=k+1}^{N}\lambda_i L\right\}. \tag{4.5}$$

4.2.3 Delimiting the Achievable Performance with Dynamic Policies

4.2.3.1 Batched Service

Determining an optimal *on-line* dynamic policy for batched service appears to be a difficult and perhaps intractable problem. For example, suppose that there is a waiting request from some client group i, when there is a remote replica j about to begin service for some batch of requests. The optimal choice between joining this batch and being served by the remote replica, versus continuing to wait for a batch to be served at the local replica, in general depends not only on the access cost c but also on the complete system state and on the statistics of the request arrival process. However, the achievable performance with dynamic policies can accurately be delimited through a combination of results for optimal *off-line* performance, with a given number of replicas and client groups, and results for optimal off-line performance in a limiting case as the number of replicas and client groups grow without bound. The off-line policies used here assume complete information about all requests made to the system (including future requests).

Consider first the problem of determining optimal off-line performance with a given number of replicas and client groups. An algorithm is developed that takes as input a request sequence (indicating both the arrival time and the client group of each request) and a maximum client delay *D*, and finds the minimum total service delivery cost for serving all of the requests in the sequence. This algorithm is based on the following observation. Define the deadline of a request as the time at which the request waiting time would equal the maximum client delay. Then, at any request deadline *t*, the minimum access cost incurred by the corresponding request is determined solely by the batch service initiations that occur within the interval [*t*–*D*, *t*], i.e., from the request arrival time to its deadline. In particular, the minimum access cost is zero if and only if the local replica begins service of a batch of requests during this interval, and otherwise is *c*.

The above observation enables the following algorithm structure. A window of duration *D* is advanced through the given request sequence, with the right endpoint of the window moving at each advance to the next request deadline. Each potential choice of batch service initiations within the current window defines a "state". When the window advances, the set of states changes, as earlier batch service initiations may now be outside of the window and some new batch service initiations may be added. Each state has an associated minimum total service delivery cost. The cost of a new state (as created when the window advances) is calculated as the minimum of the costs of the alternative prior states (before the advance of the window) that result in this new state, plus the access cost associated with the request whose deadline defines the right endpoint of the new window (according to whether or not the local replica serves a batch in the new state), plus the service cost of any new batch service initiations. When the window advances to include the deadline of the last request in the request sequence, the minimum total service delivery cost for the input request sequence and maximum client delay is given by the minimum over all current states of the associated total service delivery cost.

The feasibility of this approach depends on being able to tightly constrain the potential choices of batch service initiation times and locations, and thus the number of states associated with the current window, in a manner that still allows discovery of the minimum total service delivery cost. Assuming for clarity that no two events (request arrivals or deadlines) occur simultaneously, the constraints that are employed here are as follows:

1. A replica *may* begin service of a batch of requests at a time t, *only if* time t is a request deadline. (Otherwise, service could be postponed, with no greater service delivery cost.)
2. Replica i *may* begin service of a batch of requests at the deadline t of a client group i request, *only if* replica i did not begin service of an earlier batch of requests during the interval $(t–D, t)$. (Otherwise, the request with deadline t could have been served with the earlier batch, and the service of the remaining requests in the later batch postponed.)
3. Replica i *may* begin service of a batch of requests at the deadline t of a client group j request, for $i \neq j$, *only if* there is no replica k (k may equal i or j) that began service of a batch of requests during the interval $(t–D, t)$. (Otherwise, the request with deadline t could have been served earlier by replica k, and the service at replica i postponed, with no greater total service delivery cost.) Constraints (1)-(3) imply that each replica may begin service of a batch of requests at most once during any time period of duration D.
4. Replica i *may* begin service of a batch of requests at the deadline t of a client group j request, for $i \neq j$, *only if* there have been at least two arrivals of client group i requests in the interval $(t–D, t)$ (and that thus could belong to the batch). (Otherwise, the batch could be served by replica j instead, with no greater total service delivery cost.)
5. *Some* replica *must* begin service of a batch of requests at a deadline t, *if* there have been no batch service initiations in the interval $(t–D, t)$.
6. A replica *may not* begin service of a batch of requests at a deadline t, *if*: (a) a previous batch service initiation was at a replica i at the deadline t' of a client group i request, with $t–D < t' < t$; (b) at most $1/c$ arrivals of group i requests occurred in the interval $[t'–D, t')$; *and* (c) the batch service initiation prior to the one at time t' occurred at a time t'' with $t–D < t'' < t' < t$. (Since, in comparison to a schedule with batch service initiations at times t'', t', and t, the cost would be

no greater if the batch service initiation at time t' had not occurred, and each of the requests that belonged to the batch served at time t' were served instead with either the batch at time t'' or a batch at time t, which is possible owing to the time separation of at most D between time t'' and time t.) Essentially, this constraint says that the decision to serve the batch at time t' could be necessary in an optimal schedule (and thus the partial schedule including this batch service initiation possibly fruitful to pursue further), only if there is no batch service initiation at time t.

7. A replica *may not* begin service of a batch of requests at a time t, *if*: (a) a previous batch service initiation was at a replica i at the deadline t' of a client group i request, with $t–D < t' < t$; (b) the most recent deadline of a group i request previous to time t' occurred at a time t'' with $t–D < t'' < t' < t$; (c) at most one arrival of a group i request occurred in the interval (t'', t'); *and* (d) no batch service initiation occurred at time t'', but such a batch service initiation was not prevented by the constraints (and thus, there is a state in which replica i begins service of a batch at time t'' rather than at t'). (Since, in comparison to a schedule with batch service initiations at times t' and t but not at time t'', the cost would be no greater if each of the requests that belonged to the batch served at time t' and that arrived prior to t'' are served instead by replica i at t'', and the other requests that belonged to this batch are served instead at time t.) Essentially, this constraint says that the decision to serve a batch at replica i at time t' and not at time t'' could be necessary in an optimal schedule only if there is no batch service initiation at time t.

Although constraints (6) and (7) are somewhat more complex than the others, they can greatly reduce the number of states that need be considered. This is illustrated in Table 4.2, which shows 95% confidence intervals for the average number of states associated with the current window, and the observed maximum number, for algorithm variants using different subsets of the above constraints, with $N = 16$, $c = 0.5$, $L = 1$, $\lambda_i = 1$ for all i, and various values of the maximum client delay D.[23] For each algorithm variant and

[23] The particular algorithm implementation used for these results could accommodate 8,000,000 current states.

Table 4.2: Average and Maximum Number of States using the Optimal Offline Algorithm (N = 16, c = 0.5, L = 1, λ_i = 1 for all i)

***D* (*C*)**	**Constraints (1)-(5) only**	**Constraints (1)-(6)**	**Constraints (1)-(5), (7)**	**Constraints (1)-(7)**
0.1 (0.6526±0.0006)	6.518±0.091 2,300	4.115±0.025 69	6.141±0.073 1,285	4.045±0.024 63
0.5 (0.4645±0.0006)	2,040±190 5,072,540	98.6±1.1 3,619	666±21 349,799	86.86±0.85 2,247
1.0 (0.3999±0.0008)	- > 8,000,000	2,250±190 475,843	19,610±660 1,661,760	1,181±35 106,531
1.5 (0.3446±0.0007)	- > 8,000,000	- > 8,000,000	- > 8,000,000	13,940±460 1,342,120

value of D, 10 runs were performed, each on a different randomly generated request sequence with 25,000 request arrivals.

Although additional constraints are possible, at the cost of increased complexity in the implementation, constraints (1)-(7) were found to be sufficient to allow use of the optimal offline algorithm for a large portion of the parameter space. The algorithm can become too costly when D is large and $1/c$ is not substantially greater than $\lambda_i D$ (implying that there are many deadlines, and thus many possible batch service initiation times, within a window, and that constraint 6 becomes less effective), and/or there are a large number of replicas. Fortunately, in those cases in which the algorithm is too costly, consideration of a simple limiting case yields a lower bound on the total service delivery cost that is empirically tight. Specifically, consider the case in which there are a sufficiently large number of replicas and client groups, that whenever it would be optimal for a request to receive service from other than the local replica, there is always some batch of requests to be served at a remote replica that the request could join (without any impact on the time at which service is initiated for that batch). The minimum total service delivery cost for this case can be determined with small computational cost by a variant of the optimal offline algorithm in which each replica and its associated client group is considered in isolation, without explicit consideration of the remote replicas. Note that this lower bound on the total service delivery cost is tight not only when there is a sufficiently large number of replicas, but also when almost all requests would be served by the local replica in an optimal policy.

4.2.3.2 Fountain Service

An optimal dynamic policy for fountain service is easily determined. Consider some arbitrary point in time *t*, and denote the number of requests from client group *i* that are receiving service at time *t* by w_i. If all of the w_i are zero, no replica is dispensing service at time *t*. Otherwise, at least one replica must be serving request(s). If some replica *j* is serving request(s), then the total cost is reduced when a replica *i* ($i \neq j$) also dispenses service at time *t*, rather than requiring its local clients to receive service remotely, if and only if $w_i > 1/c$. Thus, the following policy achieves the minimum total service delivery cost for fountain service. At each time *t*, each replica *i* dispenses service if and only if either $w_i > 1/c$, or $w_i = \max_j w_j \geq 1$ and there is no $k < i$ such that $w_k = \max_j w_j$. A request from a group *i* client receives service from the local replica if it is dispensing service, and otherwise receives service from any remote replica that is dispensing service.

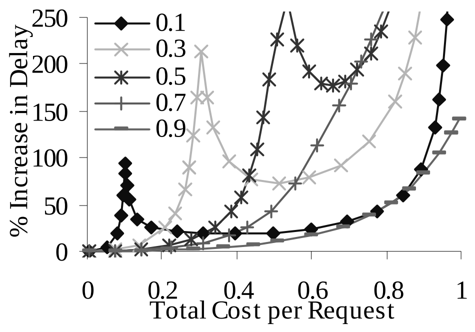
With fountain service, the maximum client delay is $D = L/r$. The total service delivery cost with the above policy, and thus a tight lower bound on the total service delivery cost achievable with a dynamic policy and fountain service, is given by

$$r\sum_{i=1}^{N}\left\{(1-p(w_i \leq 1/c)) + \sum_{k=1}^{\lfloor 1/c \rfloor} p(w_i = k)\left[kc + (1-kc)\prod_{j=1}^{i-1} p(w_j < k)\prod_{j=i+1}^{N} p(w_j \leq k)\right]\right\}, \qquad (4.6)$$

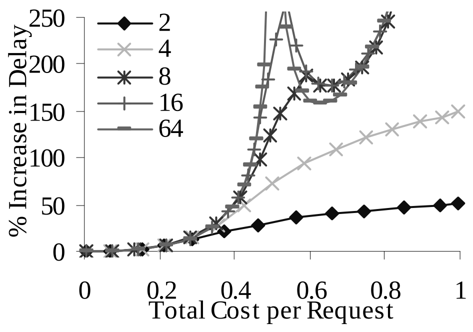
where $p(w_i = m) = \frac{(\lambda_i L / r)^m}{m!} e^{-\lambda_i L / r}$.

4.2.4 Performance Comparisons

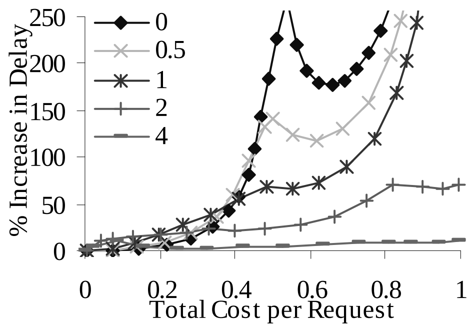
Figures 4.1 and 4.2 apply the results from Sections 4.2.2 and 4.2.3 to compare the potential performance with static versus dynamic replica selection policies, for batched and fountain service, respectively. Rather than considering the minimum total service delivery cost potentially achievable with a given maximum client delay, here (equivalently) the lowest maximum client delay potentially achievable with a given total service delivery cost is considered. Specifically, these figures show the lowest potentially achievable maximum client delay for static policies expressed as a percentage increase over that with dynamic policies, as a function of the total service delivery cost expressed as total cost per request. The total cost per request is varied by



(a) N = 16, L = 1, λ_i = 1, $c \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$



(b) L = 1, λ_i = 1, c = 0.5, $N \in \{2, 4, 8, 16, 64\}$

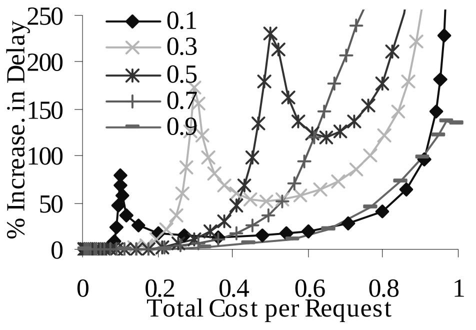


(c) N = 16, L = 1, c = 0.5, $\lambda_i = \Omega/i^{\alpha}$ with $\Omega = N / \sum_{j=1}^{N} (1/j^{\alpha})$ and $\alpha \in \{0, 0.5, 1, 2, 4\}$

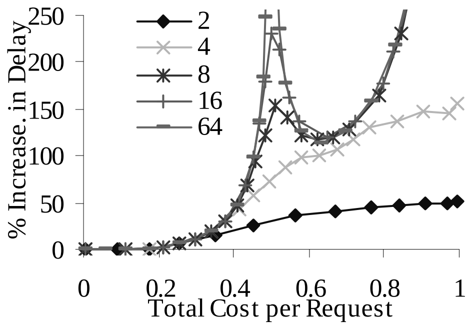
Figure 4.1: Best Potential Performance with Static Policies Relative to that with Dynamic Policies, for Batched Service.

changing the batching delay D in the case of batched service, or the service rate r in the case of fountain service. Without loss of generality, the unit of cost is chosen to be the total service required by each request, and the unit of time is chosen to be the average time between requests from a client group when the total request rate is divided evenly among the client groups. With these choices of units, $L = 1$ and $\lambda = N$. For the case of batched service and dynamic policies, the optimal offline algorithm from Section 4.2.3.1 was run on 10 randomly generated request sequences, each with 25,000 request arrivals, and the results averaged, for each set of parameters for which this algorithm was found to be feasible. For the other parameter sets, a similar methodology was followed, but using the variant of the optimal offline algorithm in which each replica and its associated client group is considered in isolation.[24]

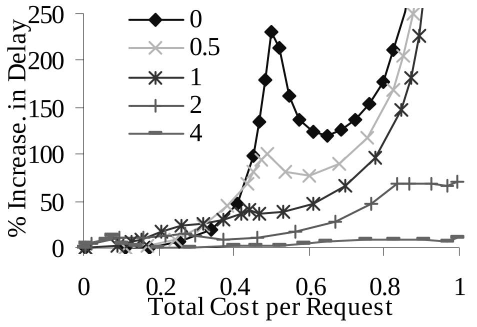
[24] In this case, owing to the relatively low execution cost, each of the 10 runs for each parameter set had 200,000 request arrivals. In general, unless using analytic expressions, all data points presented in



(a) N = 16, L = 1, λ_i = 1, $c \in$ {0.1, 0.3, 0.5, 0.7, 0.9}



(b) L = 1, λ_i = 1, c = 0.5, $N \in$ {2, 4, 8, 16, 64}



(c) N = 16, L = 1, c = 0.5, $\lambda_i = \Omega/i^{\alpha}$ with $\Omega = N / \sum_{j=1}^{N} (1/j^{\alpha})$ and $\alpha \in$ {0, 0.5, 1, 2, 4}

Figure 4.2: Best Potential Performance with Static Policies Relative to that with Dynamic Policies, for Fountain Service.

Interestingly, the results in Figures 4.1 and 4.2 are quite similar, even though they are for quite different types of service aggregation. With both batching and fountain service, dynamic policies have the potential to yield substantially better performance than static policies over large regions of the parameter space. In many cases, the lowest potentially achievable maximum client delay with static policies is over 100% higher than with dynamic policies; i.e., higher by a factor of 2.

Note the presence in many of the curves of a local maximum in the performance difference, at an intermediate value of the total service delivery cost per request. These peaks correspond to points where the optimal static policy changes between one in which all requests are served by the local replica, and one in which all requests are served by some single replica. For the cases in which all client groups have the same

Chapter 4 are generated by taking the average over 10 simulations, each simulating 200,000 or 1,000,000 requests.

request rate, the total service delivery cost per request is approximately equal to the value of the access cost *c* at these points.[25]

It is possible to obtain the asymptotic limits of the curves in Figures 4.1 and 4.2 as the cost per request (in normalized units) approaches one from the left, since in this case the batching delay *D* is so small (in the case of batched service), or the service rate *r* so large (for fountain service), that the probability that a request could receive service with more than one other request becomes negligibly small. The optimal static policy in this case is for each request to be served by the local replica. The optimal dynamic policy in this case is for each request to be served by the local replica if no previous request is waiting for service (in the case of batched service) or receiving service (in the case of fountain service) at the time of arrival of the request. In the rare event that there is such a previous request, the cost is minimized if the new request shares its service (all in the case of batched service, or whatever service remains for the previous request in the case of fountain service) with this previous request (and, in the case of fountain service, receives the remaining portion of its service locally). In Appendix B these optimal policies are analyzed, and the asymptotic limits of each curve derived. Assuming identical client group request rates these limits are $(N-1)(1-c) \times 100\%$, for both the batched and fountain service model.

Figures 4.1 and 4.2 show the *potential* performance improvements with dynamic policies, but these improvements may not be practically realizable. The next two sections consider the question of how complex a dynamic policy needs to be to achieve the full potential of this policy class.

[25] Note that these peaks occur in regions of the parameter space in which the optimal offline algorithm is feasible; only well to the left of each peak, did it became necessary to use the variant in which each replica and its associated client group is considered in isolation.

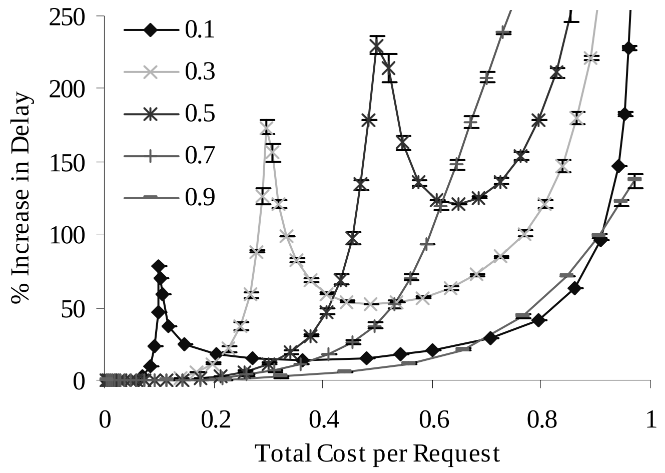


Figure 4.3: Confidence Intervals for Figure 4.2(a).

Throughout this chapter only the average values are reported; however, it should be noted that the confidence in the differences between the different policies is high. To illustrate this, Figure 4.3 shows confidence intervals capturing the true average with a confidence of 95%, using 10 simulations for each data point with identical run length to those used to generate Figure 4.2(a). Note that the confidence intervals tightly follow the shape of the curve and do not affect the results. Similar observations are true for other policies and parameter settings.

4.3 Deferred Selection vs. At-arrival Selection

A basic distinction among dynamic policies is whether replica selection occurs immediately when a request is made ("at arrival"), or whether replica selection may be deferred for some period of time. With batched service, each request is served by a single replica. This replica is selected at the request arrival time in an at-arrival replica selection policy, while with deferred selection the choice may be delayed (at most, by the maximum client delay D). With fountain service, each request immediately begins receiving service at rate b, and clients may switch replicas during their service period of duration $D = L/r$. In an at-arrival replica selection policy, a schedule giving the replica from which the client will receive service at each instant of the service period is determined at the request arrival time, while with deferred selection the replica from which a client will receive service at each time t may deferred up until time t. Note that at-arrival replica selection is a simpler approach, but deferred selection may offer the potential for improved performance, since replica selection may take into account subsequent request arrivals.

Section 4.3.1 determines optimal at-arrival replica selection policies for both batched and fountain service, and corresponding tight bounds on the achievable performance with at-arrival replica selection. Section 4.3.2 presents performance comparisons between these results and the results for general dynamic policies from Section 4.2.3.

4.3.1 Delimiting the Achievable Performance with At-arrival Policies

4.3.1.1 Batched Service

Consider first the case in which all client groups have the same request rate (λ/N). If there are one or more previous requests waiting for service by replica i when a new request from client group i arrives, the new request should join this batch. Suppose that there are no such waiting requests. Since all groups have the same request rate, in an optimal at-arrival policy a remote replica would never be selected for a newly-arriving request unless there is at least one previous request already waiting for service by that replica, and thus the next request to begin waiting for service from replica i can only be from group i. Therefore, if a remote replica is selected for the new request, the same state with respect to client group and replica i (a newly-arriving group i request, and no waiting requests at replica i) will be entered again after a time of expected duration (with Poisson arrivals) N/λ, and a cost of cL will have been incurred. If replica i is selected, the same state will be entered after a time of expected duration $D + N/\lambda$ (the expected cost is minimized if a batch is not served until time D after formation), and a cost of L will have been incurred. Comparing these two scenarios, it is optimal to select the local replica if and only if there is no remote replica with at least one waiting request and/or $(cL)/(N/\lambda) \geq L/(D+N/\lambda)$, or equivalently $c \geq 1/((\lambda/N)D+1)$; otherwise, it is optimal to select such a remote replica.

With the optimal at-arrival policy as described above, if $c \geq 1/((\lambda/N)D+1)$ all requests receive service from the local replica, and the total service delivery cost is given by $\lambda L/((\lambda/N)D+1)$. If $c < 1/((\lambda/N)D+1)$, the total service delivery cost is given by

$\lambda\left(\frac{L + cL(N-1)(\lambda/N)D}{\lambda D + 1}\right)$, where the term in parentheses gives the expected cost per request, as computed by dividing the expected total cost to serve all of the requests in a batch by the expected number of requests in a batch.

Consider now the general case in which client groups may have differing request rates. Suppose that when a client group i request arrives there are no previous requests waiting for service by any replica. Recalling that replicas/groups are indexed from 1 to N in non-increasing order of the group request rates, analogously to the optimal static policy there is an optimal index k ($1 \le k \le N$), such that for $i \le k$, the new request begins a batch that will receive service by the local replica, while for $i > k$, the new request begins a batch that will receive service by replica 1.

For client groups i with $2 \le i \le k$, the optimal replica selection policy is as described in the case of homogeneous groups, but with the condition $c \ge 1/((\lambda/N)D+1)$ replaced by $c \ge 1/(\lambda_i D+1)$. For group i requests with $i > k$, it is optimal to select a remote replica that already has a waiting request (if any), and otherwise to select replica 1. Finally, consider group 1 requests. If there is at least one previous request that is waiting for service by replica 1, or if there are no previous requests waiting for service by any replica, it is optimal to select replica 1. The case in which there are no previous requests that are waiting for service by replica 1, but at least one request waiting for service by some remote replica, is more complex than with homogenous groups, however, when $k < N$. This is since requests from other than group 1 may initiate new batches to be served by replica 1, which increases the desirability of selecting replica 1 in this case. Note though, that when $k < N$ it must be true that $1+\lambda_i D < \lambda_1 D$ for some client group i (namely, each group i for $i > k$), since it can only be desirable for a group i request to begin a new batch to be served by replica 1 rather than by replica i if the expected number of group 1 requests that will be served in that batch exceeds the expected number of group i requests. This implies that $\lambda_1 D > 1$, and therefore that $c \ge 1/(\lambda_1 D+1)$ for $c \ge 1/2$. Since it is even more desirable than with homogeneous groups to select replica 1 for a newly-arriving request from group 1, in the event that there are no previous requests that are waiting for service by replica 1 but at least one request waiting for service by some remote replica, for $c \ge 1/2$ (and $k < N$) it must be true that is optimal to select replica 1 in this case. For the results shown in Section 4.3.2 for client groups with differing request rates, the access cost is chosen as $c = 1/2$, and simulation is used to determine the optimal index k and the minimum total service delivery cost according to the optimal policy as described above.

4.3.1.2 Fountain Service

With fountain service, the replica from which a newly-arriving request will receive service must be determined for each instant of the client's service period of duration $D = L/r$. Denoting the time since the previous request arrival by t_a, it is necessary to determine: (1) which replica should be scheduled to provide service for the last $\min[t_a, L/r]$ of the service period, and (2) for each time instant at which the local replica is not scheduled to dispense service during the initial $\max[0, L/r–t_a]$ of the service period, whether the local replica should now be scheduled for that time, or whether the new request should receive service from a remote replica already scheduled (such a replica must exist owing to the service period of the previous request).

Consider first the case in which all client groups have the same request rate λ/N. In this case, it is clearly optimal to schedule the local replica to dispense service for the portion of the service period during which no replica is already scheduled (the last $\min[t_a, L/r]$). For each time offset t at which the local replica is not scheduled to dispense service, within the initial $\max[0, L/r–t_a]$ of the service period, it is optimal to schedule the local replica, rather than to have the new request receive service from a (remote) replica already scheduled, if and only if $r \leq rc(1+(\lambda/N)t)$, or equivalently $t \geq N(1–c)/(c\lambda)$. Denoting the threshold value $\min[N(1–c)/(c\lambda), L/r]$ by T, the above observations yield the following tight lower bound on the total service delivery cost achievable with an at-arrival replica selection policy and fountain service:

$$r\left\{N\left(1-e^{-(\lambda/N)(L/r-T)}\right)+\left(e^{-(\lambda/N)(L/r-T)}-e^{-\lambda(L/r-T)}\right)c\lambda T \right.$$
$$\left. +e^{-\lambda(L/r-T)}\left(\left(1-e^{-\lambda T}\right)+c\left(\lambda T-\left(1-e^{-\lambda T}\right)\right)\frac{N-1}{N}\right)\right\}. \qquad (4.7)$$

This expression is derived as follows. Consider the state of the system at an arbitrary point in time under the operation of an optimal at-arrival policy. If there was at least one request from client group i at an offset prior to the current time in the interval $[–L/r$,

–*T*], replica *i* will currently be dispensing service, yielding the first term within the outer parentheses. If there were no requests from group *i* but at least one request from some other group *j* in [–*L*/*r*, –*T*] (and thus replica *j* is currently dispensing service), any group *i* requests that were made in [–*T*, 0] will currently be receiving service from a remote replica, yielding the second term within the outer parentheses. Finally, if there were no requests from any group in [–*L*/*r*, –*T*], one replica will currently be dispensing service if and only if at least one request arrived in the interval [–*T*, 0], and all requests that arrived in the interval [–*T*, 0] from client groups other than that from which the first such request arrived, will currently be receiving service from a remote replica.

For the general case in which client groups may have differing request rates, the optimal choice between receiving service from a remote replica already scheduled, or scheduling the local replica, is determined according to the time offset *t* from the beginning of the service period as in the case of homogeneous client groups. For $t \geq T_i = \min[(1-c)/(c\lambda_i), L/r]$, it is optimal to schedule the local replica; otherwise, it is optimal to receive service from the remote replica.

Unlike in the case of homogeneous groups, for new requests from other than group 1 (that with the highest request rate) it may not be optimal to schedule the local replica for the portion of the service period of a new request during which no replica is already scheduled. Consider a newly-arriving request from other than group 1, and a time offset $t > T_1$ from the beginning of the service period, at which no replica is already scheduled. In this case, it is optimal to schedule the local replica to dispense service at time *t* if and only if $r(1-e^{-\lambda_1(t-T_1)})+e^{-\lambda_1(t-T_1)}rc\lambda_1 T_1 \leq rc(1+\lambda_i t)$; otherwise, replica 1 should be scheduled instead. This relation takes into account the possibility that even if the local replica is scheduled to dispense service at time *t*, replica 1 may also be so scheduled at a subsequent group 1 request arrival. Consider now $t \leq T_1$. It is optimal to schedule the local replica to dispense service at time *t* if and only if $rc\lambda_1 t \leq rc(1+\lambda_i t)$; otherwise, it is optimal to schedule replica 1 instead. Combining these two cases yields the condition

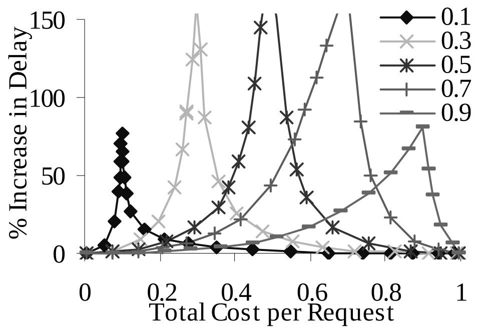
$$\left(1-e^{-\lambda_1(t-\min[t,T_1])}\right)+e^{-\lambda_1(t-\min[t,T_1])}c\lambda_1 \min[t,T_1] \leq c(1+\lambda_i t)\,. \tag{4.8}$$

It is straightforward to verify that this condition divides the interval [0, *L*/*r*] into (at most) three regions: an initial region in which the condition holds, a second region that may or may not be present and in which the condition does not hold, and (should the second region be present) a third region that may or may not be present in which the condition again holds. Define T_i' and T_i'' to be the boundary values separating the regions, should all three exist, with $T_i'' < T_i'$; note that $T_i' < T_i$ in this case. If just the first two regions exist, then $T_i = L/r$. In this case define T_i'' as the boundary value separating these regions and define $T_i' = T_i$ (= L/r). Finally, if only the first region exists, define T_i'' = $T_i' = T_i$. Then it is optimal to schedule the local replica to dispense service at t for $t \leq T_i''$ and for $t \geq T_i'$, and replica 1 for $T_i'' < t < T_i'$.

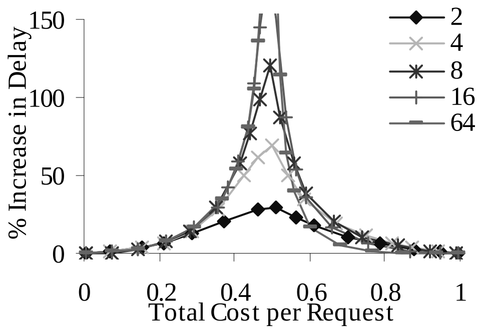
Defining f_i as the fraction of requests that are from client group i, analysis of the above optimal policy yields the following tight lower bound on the total service delivery cost achievable with an at-arrival policy and fountain service, for the general case of heterogeneous client groups:

$$\begin{aligned}
& r\sum_{i=2}^{N}\Big\{\left(1-e^{-\lambda_i(L/r-T_i)}\right)+\left(e^{-\lambda_i(L/r-T_i)}-e^{-\lambda(L/r-T_i)}\right)c\lambda_i T_i+e^{-\lambda(L/r-T_i)}(1-f_i)c\left(\lambda T_i-\left(1-e^{-\lambda T_i}\right)\right)f_i \\
& +\left(\left(e^{-\lambda(L/r-T_i)}-e^{-\lambda(L/r-T_i')}\right)+\left(e^{-\lambda(L/r-T_i'')}-e^{-\lambda L/r}\right)\right)f_i \\
& +e^{-\lambda(L/r-T_i')}f_i c\left(\lambda_i(T_i'-T_i'')+\left(1-e^{-\lambda(T_i'-T_i'')}\right)(1-f_i)\right)+\left(e^{-\lambda(L/r-T_i')}-e^{-\lambda(L/r-T_i'')}\right)f_i c\lambda_i T_i'' \\
& +e^{-\lambda(L/r-T_i')}e^{-\lambda_1(T_i'-T_1)}\left(1-e^{-(\lambda-\lambda_1)(T_i'-\max[T_1,T_i''])}\right)(\lambda_i/(\lambda-\lambda_1))(1-c\lambda_1 T_1) \\
& +\left(e^{-\lambda(L/r-\max[T_1,T_i''])}-e^{-\lambda(L/r-T_i'')}\right)f_i\left(1-c\left(\lambda_1\max[T_1,T_i'']+\left(\lambda\max[0,T_1-T_i'']/\left(1-e^{-\lambda\max[0,T_1-T_i'']}\right)-1\right)f_1\right)\right)\Big\} \\
& +r\Big\{\left(1-e^{-\lambda_1(L/r-T_1)}\right)+\left(e^{-\lambda_1(L/r-T_1)}-e^{-\lambda(L/r-T_1)}\right)c\lambda_1 T_1+\left(e^{-\lambda(L/r-T_1)}-e^{-\lambda L/r}\right)f_1 \\
& +e^{-\lambda(L/r-T_1)}(1-f_1)c\left(\lambda T_1-\left(1-e^{-\lambda T_1}\right)\right)f_1\Big\}. \qquad (4.9)
\end{aligned}$$

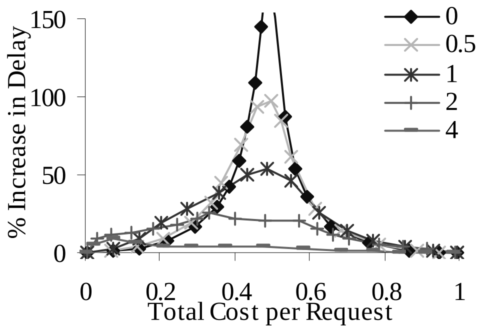
A derivation of this expression is presented in Appendix C.



(a) N = 16, L = 1, λ_i = 1, $c \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$



(b) L = 1, λ_i = 1, c = 0.5, $N \in \{2, 4, 8, 16, 64\}$

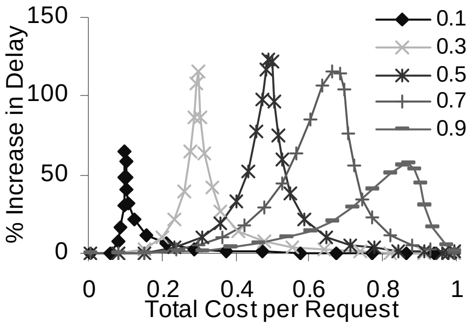


(c) N = 16, L = 1, c = 0.5, $\lambda_i = \Omega/i^{\alpha}$ with $\Omega = N / \sum_{j=1}^{N} (1/j^{\alpha})$ and $\alpha \in \{0, 0.5, 1, 2, 4\}$

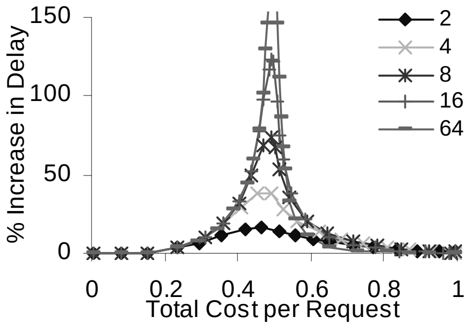
Figure 4.4: Best Potential Performance with At-arrival Policies Relative to that with General Dynamic Policies, for Batched Service.

4.3.2 Performance Comparisons

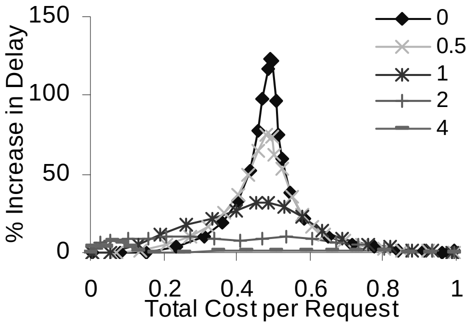
Figures 4.4 and 4.5 apply the results from Sections 4.3.1 and 4.2.3 to compare the potential performance with at-arrival versus general dynamic replica selection policies, for batched and fountain service, respectively. As illustrated in the figures, use of deferred selection can potentially yield substantial performance improvements, by a factor of two or more in some cases, although only for fairly narrow ranges of model parameter values. In particular, large potential performance improvements are seen only when the total cost per request is approximately the same as the access cost when a request is processed (entirely) remotely, equal to cL. In such regions, the potential performance improvements are maximized as the client groups become more homogeneous, as the number of replicas and client groups increases, and for values of c (in normalized units) between 0.3 and 0.7. Note that in the case of homogeneous client groups, as the total cost per request decreases (i.e., the batching delay D for batched



(a) $N = 16$, $L = 1$, $\lambda_i = 1$, $c \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$



(b) $L = 1$, $\lambda_i = 1$, $c = 0.5$, $N \in \{2, 4, 8, 16, 64\}$



(c) $N = 16$, $L = 1$, $c = 0.5$, $\lambda_i = \Omega/i^{\alpha}$ with $\Omega = N / \sum_{j=1}^{N} (1/j^{\alpha})$ and $\alpha \in \{0, 0.5, 1, 2, 4\}$

Figure 4.5: Best Potential Performance with At-arrival Policies Relative to that with General Dynamic Policies, for Fountain Service.

service increases, or the service rate *r* for fountain service decreases), the point at which the total cost per request equals *cL* is exactly (for batched service) or approximately (for fountain service) the point at which the optimal at-arrival policy changes from one in which all requests receive all service from the local replica, to one in which some service is received remotely.

4.4 Local State vs. Global State

Dynamic replica selection policies use information about the current system state. A key part of this information concerns what requests are waiting for service (in the case of batched service) or receiving service (in the case of fountain service). Here, “local state” policies are defined as those that make replica selection decisions for client group *i* requests, and service scheduling decisions for replica *i*, based on the currently outstanding group *i* requests. “Global state” policies, in contrast, are defined as those that use information concerning the current requests from other client groups, in addition to the local group. Note that both types of policies may also use other types of state information, in particular concerning the times at which replicas are scheduled to serve batches (in the case of batched service) or which replicas are currently dispensing service (in the case of fountain service).

Section 4.4.1 describes candidate local state replica selection policies for batched and fountain service. Section 4.4.2 presents a candidate on-line global state policy for batched service. Section 4.4.3 compares performance with these policies to the limits on achievable performance from Section 4.2.3.

4.4.1 Candidate Local State Policies

4.4.1.1 Batched Service

In the candidate local state policy for batched service, replica selection uses the following two rules. First, when a replica *i* initiates service of a batch, all currently waiting client group *i* requests receive this service. Second, when a remote replica initiates service of a batch at a time *t*, a waiting client group *i* request that arrived at a time t_a receives this service *if*: (a) for the earliest waiting group *i* request with arrival time $t_a' \leq t_a$, there are fewer than $1/c - \lambda_i(t_a'+D-t)$ waiting group *i* requests with arrival time no earlier than t_a'; *and* (b) there is no batch service initiation that has been scheduled (by time *t*) at any replica within the time interval $(t, t_a+D]$.[26]

A batch service initiation is scheduled (for a time possibly in the future) at a replica *i* whenever one of the following events occurs: (a) the waiting time of a request from client group *i* reaches the maximum duration *D*; (b) a request arrives from group *i*, and the number of group *i* requests that are waiting for service reaches at least $1/c$; *or* (c) a request arrives from group *i* when there is no future batch service initiation that has been scheduled at any replica, and the number of requests from group *i* that are waiting for service reaches at least $\max[(2/3)(1/c), 2]$. The motivation for scheduling a batch at the last of these events is to increase the likelihood that when batches are served that

[26] No significant advantage have been observed by policies that treat each requests individually, allowing some subset of outstanding requests, local to a particular replica, to retrieve service, while others defer their decision. For example, this local state policy gives essentially the same results as if the above rule is modified such that rule (a) require there to be fewer than $1/c - \lambda_i(t_a'+D-t)$ waiting group *i* requests for *each* waiting group *i* request with arrival time $t_a' \leq t_a$, rather than only for the *first* such request.

have less than $1/c$ requests from any one client group, the replica that serves the batch is one with a relatively large number of requests from its local client group.

When a replica i schedules a batch service initiation, the time of this service initiation is chosen as t_a+D, where t_a denotes the arrival time of the earliest request among the client group i requests that are waiting for service, *if*: (a) there is a future batch service initiation that has been scheduled by some other replica; (b) the most recent batch service initiation was by replica i; *or* (c) the most recent batch service initiation occurred later than time t_a. Otherwise, the time of the batch service initiation is chosen as the maximum of the current time, and $t_{last} + D$, where t_{last} denotes the time of the last batch service initiation at any replica.

4.4.1.2 Fountain Service

In the candidate local state policy for fountain service, a request receives service from the local replica whenever that replica is dispensing service during the service period of the request, and otherwise receives service from a remote replica. So as to provide a tunable upper bound on the number of replicas from which a request receives service, and on the frequency with which a replica initiates/terminates service, whenever a replica i initiates service it is constrained to continue this service for at least L'/r, where L' ($L' \leq L$) is a protocol parameter, or until there are no requests from client group i that are receiving this service.

Replica i initiates service whenever one of the following events occurs while the replica is not already dispensing service: (a) a request arrives from client group i, and no replica is currently dispensing service; (b) the only replica dispensing service is terminating its service, there is at least one group i request for which further service is required, and no other replica initiates service upon this service termination;[27] (c) a request arrives from group i, and the new number of outstanding group i requests is at least $1/c$, as is the expected average number of group i requests that will be receiving service over the next L'/r;[28] *or* (d) a request arrives from group i while there is only one

[27] If there are requests from multiple client groups for which further service is required, multiple replicas may be eligible to initiate service in this scenario, only one of which should actually do so (selection of which may be random, or according to some deterministic rule).

[28] This latter quantity can be efficiently calculated by keeping track of the sum, over all outstanding client group i requests, of the service that each will receive over the next L'/r; denoting this sum by S_Σ, the

replica that is dispensing service, this replica has been dispensing service for at least L'/r, it has been at least L'/r since replica i was last dispensing service, and the new number of outstanding group i requests, as well as the expected average number of group i requests that will be receiving service over the next L'/r, is at least $\max[(2/3)(1/c), 2]$.

A replica i terminates its service whenever one of the following events occurs while it is dispensing service: (a) a group i client stops receiving service, and there are no remaining group i clients receiving service; (b) a group i client stops receiving service, there are less than $1/c$ remaining group i clients receiving service, and the replica has been dispensing service for at least L'/r; *or* (c) some other replica initiates service, there are less than $1/c$ group i clients receiving service, and the replica has been dispensing service for at least L'/r.

4.4.2 Candidate On-line Global State Policies

To give additional intuition for the potential performance differences between at-arrival policies and dynamic policies, as well as between global state and local state policies, this section considers the performance achieved by *on-line* global state policies. Note that the optimal dynamic policy for fountain service, defined in Section 4.2.3.2, is in fact an optimal on-line global state policy. Therefore, this sub-section only defines a policy for the batched service model.

Unlike local state policies, global state policies have knowledge of the outstanding requests of all replicas. Taking advantage of this knowledge allows these policies to better determine which replica should serve a batch at times when a deadline is reached. As for the local state policy (defined in Section 4.4.1.1), no significant differences have been observed for policies that treat requests on an individual basis, versus policies which treat requests on a per replica basis. The policy presented here assumes that either all or no outstanding requests local to a particular replica is served by a batch. Further, a batch is only served when some outstanding request reaches its deadline.

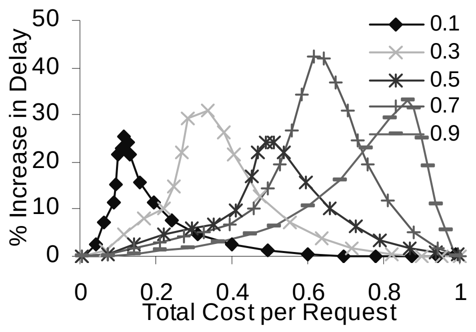
expected average number of client group i requests that will be receiving service over the next L'/r, assuming Poisson request arrivals, is given by $((S_\Sigma/r) + \lambda_i(L'/r)^2/2)/(L'/r)$.

Assuming replica i initiates service of the batch at time t and the longest outstanding client request waiting from some other client group j arrived at a time t_j, then (i) all currently waiting client group i requests receive this service, *and* (ii) so does all of replica j's requests *if* for each client group k with $t_k \leq t_j$ (including j) there are fewer than $1/c - \lambda_k(t_k+D-t)$ waiting group k requests, where t_k is the arrival time of that client groups' most outstanding request. While these rules ensure that replicas with more local requests are more likely to have requests reaching deadlines, these rules do not eliminate the chance of a replica with less than $1/c$ local request reaching a deadline.

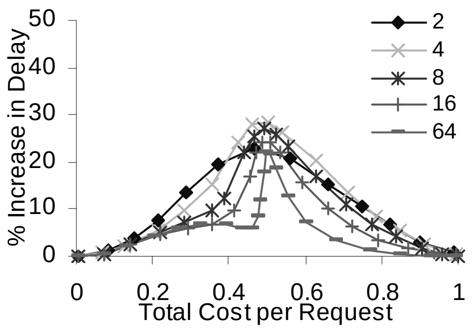
When a deadline is reached, the local replica serves the batch if it has at least $1/c$ local requests; otherwise, there are two natural candidates to serve the batch: (i) the replica that has the next deadline among the set of replicas with at least $1/c$ local requests, or (ii) the replica with the most local request among the set of all replicas with requests that will obtain service at this deadline. Since the replica with the current deadline is always included in at least the second of these two sets, at least one natural candidate always exists.

In an attempt to select the better of these two candidates, the policy used here weighs the expected cost of these two candidates. To do this, note that the benefit of initiating service at the first replica associated with group (i) is a reduction in the amount of batches served. While only one of the two batches are used (allowing a reduction in cost with equal to L), it is important to note that there is an indirect cost associated with having a replica with at least $1/c$ local requests serve the batch before its next deadline. This cost is assumed to grow linearly with time, and (as a first order approximation) the cost per time unit (that the batch is moved earlier) is approximated with the average service delivery cost per replica (C/N).[29] The benefit of initiating service at this replica can hence be approximated by $L - \delta_{(i)}C/N$, where $\delta_{(i)}$ is the time until this replica's next deadline. This benefit must be compared against the additional cost associated with all $n_{(ii)}$ of the request local to the replica with the most local request of the replicas in group (ii) retrieving service remotely. Comparing the benefits of fewer batches (i.e., L –

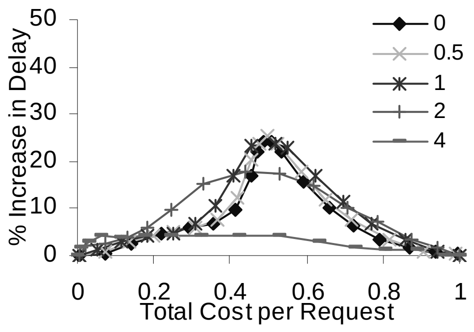
[29] In a real system the total delivery cost (per time unit) C can be obtained using some form of on-line estimation technique (e.g., an exponentially weighted moving average). For the results presented here, binary search over long-duration simulations was used to find a correct value for C.



(a) $N = 16, L = 1, \lambda_i = 1, c \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$



(b) $L = 1, \lambda_i = 1, c = 0.5, N \in \{2, 4, 8, 16, 64\}$



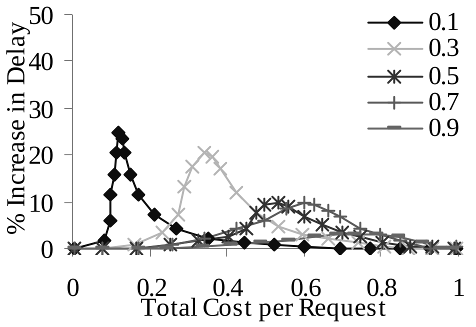
(c) $N = 16, L = 1, c = 0.5, \lambda_i = \Omega/i^\alpha$ with $\Omega = N / \Sigma_{j=1}^{N}(1/j^\alpha)$ and $\alpha \in \{0, 0.5, 1, 2, 4\}$

Figure 4.6: Performance with Local State Policy Relative to the Best Potential Performance with General Dynamic Policies, for Batched Service.

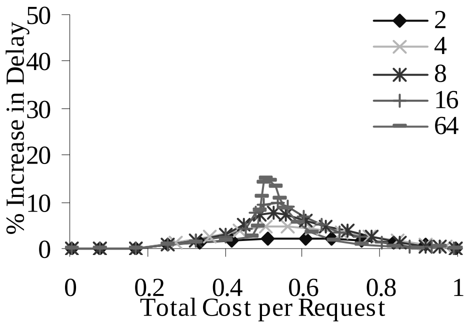
$\delta_{(i)}C/N$) against the remote access cost associated with these $w_{(ii)}$ requests retrieving service remotely (i.e., $cw_{(ii)}L$), the policy initiates service at the first replica in group (i) if the benefit is greater than the costs, otherwise the policy initiates service at the replica with the most local request in group (ii).

4.4.3 Performance Comparison

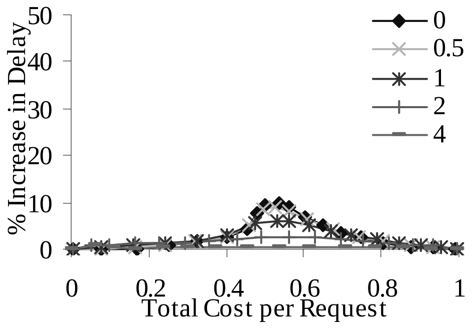
Figures 4.6 and 4.7 compare the performance of the candidate local state policies described in Section 4.4.1, as evaluated using simulation, to the best potential performance with general dynamic policies, determined as described in Section 4.2.3, for batched and fountain service, respectively. The results for fountain service, as shown in Figure 4.7, are the easiest to interpret. Here the local state policy (with $L' = L/2$) achieves within 25% of the lowest potentially achievable maximum client delay in all cases considered. Thus, in the context of fountain service, although there are



(a) $N = 16, L = 1, \lambda_i = 1, c \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$



(b) $L = 1, \lambda_i = 1, c = 0.5, N \in \{2, 4, 8, 16, 64\}$

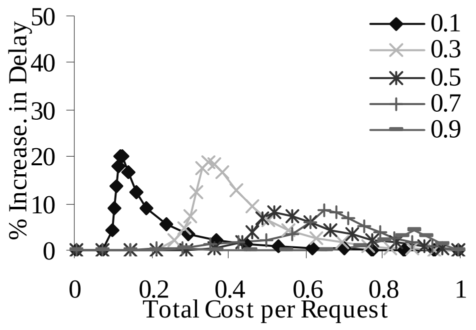


(c) $N = 16, L = 1, c = 0.5, \lambda_i = \Omega/i^{\alpha}$ with $\Omega = N / \sum_{j=1}^{N} (1 / j^{\alpha})$ and $\alpha \in \{0, 0.5, 1, 2, 4\}$

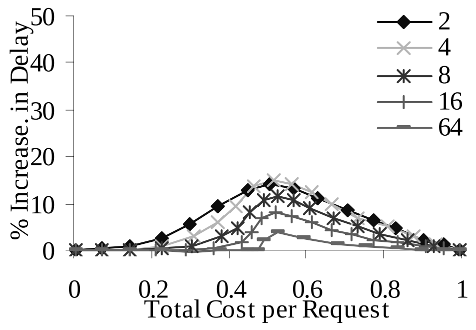
Figure 4.7: Performance with Local State Policy Relative to the Best Potential Performance with General Dynamic Policies, for Fountain Service ($L' = L/2$).

substantial potential performance benefits in using dynamic rather than static replica selection policies (as shown in Section 4.2), and deferred rather than at-arrival policies (as shown in Section 4.3), within the class of deferred, dynamic policies, simpler local (vs. global) state policies appear able to achieve close to optimal performance.

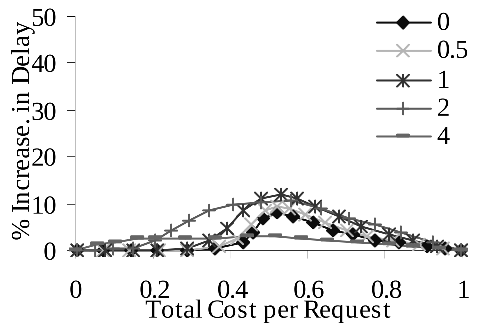
The results for batched service, as shown in Figure 4.6, are complicated by the fact that the best potential performance is delimited using the optimal *off-line* performance. It is uncertain as to what portion of the performance gaps shown in Figure 4.6 are owing to use of local state vs. global state, and what portion are owing to use of on-line vs. off-line policies. Figure 4.8 illustrates the performance difference between the above local state policy and the candidate (on-line) global state policy, defined in Section 4.4.2. Note that these results are much more similar to the fountain service results, presented in Figure 4.6. Based on these results, it is conjectured that the performance gaps between the candidate local state policy and the optimal *on-line*



(a) N = 16, L = 1, λ_i = 1, $c \in$ {0.1, 0.3, 0.5, 0.7, 0.9}



(b) L = 1, λ_i = 1, c = 0.5, $N \in$ {2, 4, 8, 16, 64}



(c) N = 16, L = 1, c = 0.5, $\lambda_i = \Omega/i^{\alpha}$ with
$\Omega = N / \sum_{j=1}^{N}(1/j^{\alpha})$ and $\alpha \in$ {0, 0.5, 1, 2, 4}

Figure 4.8: Performance with Local State Policy Relative to the Performance with the Candidate Global State Policy, for Batched Service.

performance are intermediate in size to those for fountain service (Figure 4.6) and those shown in Figure 4.7, and are closer in size to the former than to the latter.

Performance comparisons among all of the considered policy classes are presented in Figure 4.9, which show various examples of parameter settings, for the batched service model.

4.5 Average Delay Batching Policy Comparison

While previous sections of this chapter have compared policy classes with regards to the maximum delay metric D, this section compares policy classes with regards to the average delay metric A. Note that for the case of fountain service $A = D$; thus, for the fountain service model, all results are the same as for the maximum delay metric. For the batched service model, comparing policy classes with regards to the average delay metric (rather than with regards to the maximum delay metric) is a much more complex task, especially in environments with multiple client groups and replica sites. For example, some policy may achieve a low average delay by giving preferential treatment to some group of clients; however, such policy can not be fairly compared with policies that provide each client group with the same average delay. To compare policies under relatively fair circumstances, this section requires some form of fairness to be taken into consideration; specifically, policies are compared that provide each client group with equal (or roughly equal) average delays.

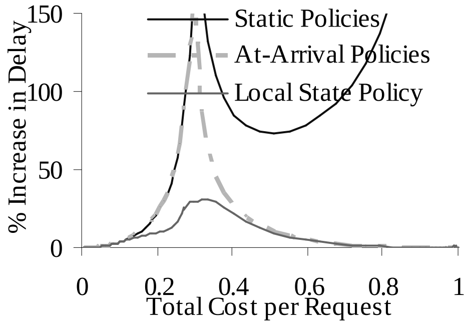
As it appears much more complex to delimiter the performance of each policy class, Section 4.5.1 defines an optimal static policy (delimiting the performance using static policies), Section 4.5.2 defines a candidate dynamic policy using global state information (providing an idea of achievable performances of dynamic on-line global state policies), and Section 4.5.3 defines a candidate at-arrival policy (providing an idea of achievable performances of at-arrival on-line policies). With the exception of the dynamic global state policy (which achieves fairness for the homogenous case, as well as the special cases when all requests always are served locally, or by the replica with the most local requests, respectively), all policies provide all client groups with the same average delays. Section 4.5.4 compares the relative performance of these policies, and relates these results to the maximum delay results, obtained in previous sections of this chapter.

4.5.1 Delimiting the Achievable Performance with Static Policies

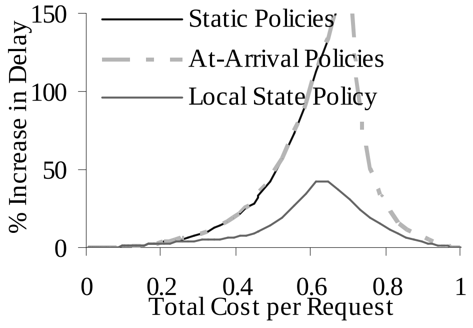
Using the same arguments as used in Section 4.2.2, in an optimal static policy there is an index k ($1 \le k \le N$), such that all requests from group i clients, for $i \le k$, are served by the local replica, while all requests from group j clients, for $j > k$, are served by replica 1. Ensuring that each client (or group of clients) has the same expected time until service (i.e., the same average delay A), independent of its geographic location or which replica serves a group of client requests, the static optimal cost can be calculated as,

$$\min_{k=1,2,\cdots,N}\left\{\frac{L}{(n_1+f_1)/\left(\lambda_1+\sum_{i=k+1}^{N}\lambda_i\right)}+\sum_{i=2}^{k}\frac{L}{(n_i+f_i)/\lambda_i}+c\sum_{i=k+1}^{N}\lambda_i L\right\}, \tag{4.10}$$

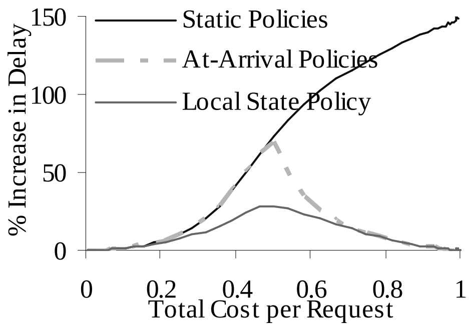
where n_i and f_i are selected such that the average delay A, for clients receiving service from replica i, is equal to some target delay A^*, calculated as



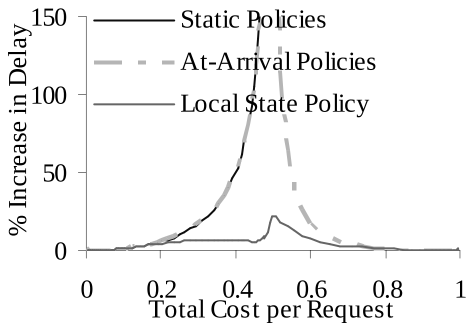
(a) $c = 0.3$, $N = 16$, equal request rates



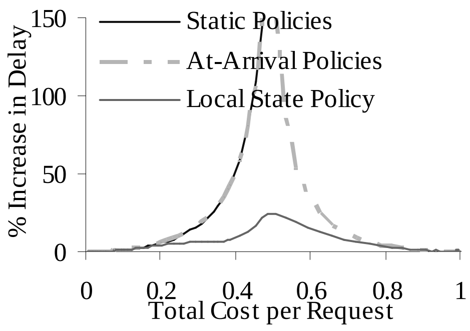
(b) $c = 0.7$, $N = 16$, equal request rates



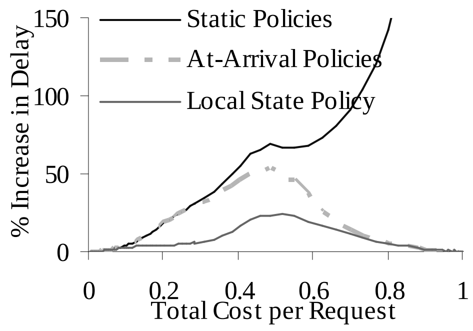
(c) $c = 0.5$, $N = 4$, equal request rates



(d) $c = 0.5$, $N = 64$, equal request rates



(e) $c = 0.5$, $N = 16$, equal request rates



(f) $c = 0.5$, $N = 16$, $\lambda_i = \Omega/i$

Figure 4.9: Best Potential Performance with Static Policies and with At-arrival Policies, and Actual Performance with a Local State Policy, Relative to the Best Potential Performance for General Dynamic Policies, for Batched Service.

$$\frac{n_i\left(n_i+2f_i-1\right)}{2\left(\lambda_i+\partial_{i=1}\sum_{j=k+1}^{N}\lambda_j\right)\left(n_i+f_i\right)}, \qquad (4.11)$$

where $\partial_{i=1} = 1$ if $i = 1$ and 0 otherwise.

4.5.2 A Candidate Dynamic Global State Policy

Consider first the case in which all client groups have the same request rate. With the candidate policy presented here, each request associated with a replica receives service with the current batch *if* the local replica is the replica initiating service, *or* the local replica has less than n' outstanding requests, where n' is chosen to be equal to an integer parameter n with probability $1-f$, and $n+1$ with probability f. (Using the protocol parameters n and f ($0 \leq f < 1$) the threshold n' can be probabilistically determined at times when there less than n outstanding local requests.) Note that n' greater than $1/c$ is sub-optimal, as the cost associated with serving these n' requests would be smaller if served locally.

Using the above rule to determine which clients should be served by a (potential) batch, the policy initiates service (of a new batch), at the replica with the most local requests, as soon as there are a total of m' requests that would retrieve service with the batch, where m' is equal to an integer parameter m with probability $1-g$, and $m+1$ with probability g. Using the protocol parameters m and g ($0 \leq g < 1$) the threshold m' can be probabilistically determined at times when there are less than m requests that would retrieve service with the batch. Note that $n' = 1$ corresponds to the extreme cases where all service is retrieved locally, which is optimal when the service cost is much smaller than the remote access cost. Similarly, $n' \to \infty$ corresponds to the extreme cases where all outstanding requests in the system are served simultaneously by the replica with the most local requests, which is optimal when the remote access cost is much smaller than the service cost.

To extend the above policy to the general case, the two parameters A^* and y are used. A^* can be considered as a target delay, while y is a weighing factor used to ensure that replica sites with higher arrival rates are less likely to listen remotely, given the same number of accumulated local requests. For each replica i, m_i and g_i are chosen such that $A^* = \frac{m_i(m_i + 2g_i - 1)}{2\lambda_i(m_i + g_i)}$, and n_i and f_i are chosen such that $n_i + f_i = \max[1, \lceil 1/c \rceil - y\lambda_i]$.

Selecting n_i' as above, the same rule can be used to determine which requests should receive service with a batch. However, in contrast to the homogenous case a somewhat more general rule is used to determine when a batch should be initiated. With batches only being scheduled immediately following a request, a new service period is scheduled *if* (i) there would be a total of $\max_{i=1..N} m_i'$ requests that would receive service with the batch, *or* (ii) a replica reaches max[m_i', n_i'] local requests. For the first case, the replica with the most local requests initiates service (with ties giving preference to the replica with the lowest arrival rate). For the second case, service is initiated at the replica reaching its threshold. Note that the above policy achieves the same average delays for all client groups whenever (i) all groups have the same request arrival rate, (ii) all service is retrieved locally, or (iii) requests always are served by the first initiated service batch, independent of replica.[30]

4.5.3 A Candidate At-arrival Policy

This section defines a policy that is similar in nature to the at-arrival maximum delay batching policy, presented in Section 4.3.1.1. The policy uses two thresholds k_1 and k_2 ($0 \leq k_1 \leq k_2 \leq N$) to split the replicas into three categories. The k_1 replicas with the highest arrival rates are in category one, the following k_2–k_1 replicas are in category two, and the remaining N–k_2 replicas (with the lowest arrival rates) are in category three. Based on this classification, (i) requests associated with replicas in the first category are always served locally, (ii) requests associated with replicas in the second category are always served by the replica within this category, which first receives a local request, following the previous service initiation at some replica in this category, and (iii) requests associated with a replica of the third category are always served by replica 1. Note that $k_1 = k_2 = N$ captures the case when all requests are served locally, and $k_1 = 0$, $k_2 = N$ captures the case when requests are always served by the first replica receiving a request, after the previous service initiation. Further, under the best parameter settings (i.e., the best choice of k_1 and k_2), replicas with sufficiently large arrival rates will always serve (at least) its local requests, and client requests associated with replicas with

[30] While the policy does not guarantee fairness among heterogeneous replicas and client groups, similar performance improvements can be obtained using a fair deferred policy that split the replicas in three classes, based on their arrival rates, much like the at arrival policy defined in Section 4.5.3. However, this policy is more restrictive and does not achieve quite as good performance for the homogenous case, as the policy presented here.

sufficiently low arrival rates will be served by a replica that is likely to have the most local requests.

As with the static optimal policy, the parameters of this policy are chosen such that all groups of clients experience the same average delay, independent of their location (or which request they are served). Using the above classification and a target delay A^*, a replica in category one (i.e., $1 \le i \le k_1$) initiates service if this batch would serve n_i (with a probability $1-f_i$) or n_i+1 (with a probability f_i), where n_i and f_i are chosen such that the target delay A^* is equal to the expected average delay

$$\frac{n_i(n_i + 2f_i - 1)}{2(n_i + f_i)\left(\lambda_i + \partial_{i=1}\sum_{j=k_2+1}^{N}\lambda_j\right)}, \tag{4.12}$$

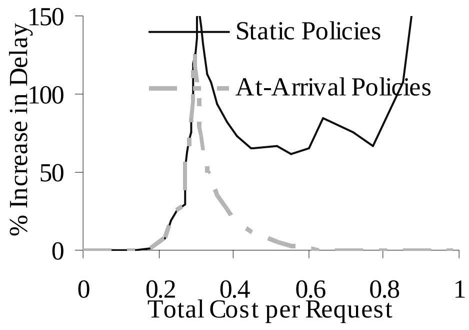
where $\partial_{i=1} = 1$ if $i = 1$, and 0 otherwise. Similarly, in category two (i.e., replicas such that $k_1 < i \le k_2$) the first replica to receive a request (after the previous service initiation in this category) initiates service as soon as it would serve n_g (with a probability $1-f_g$) or n_g+1 (with a probability f_g), where n_g and f_g are chosen such that the target delay A^* is equal to the expected average delay

$$\frac{n_g(n_g + 2f_g - 1)}{2(n_g + f_g)\left(\sum_{j=k_1+1}^{k_2}\lambda_j + \partial_{k_1=0}\sum_{j=k_2+1}^{N}\lambda_j\right)}, \tag{4.13}$$

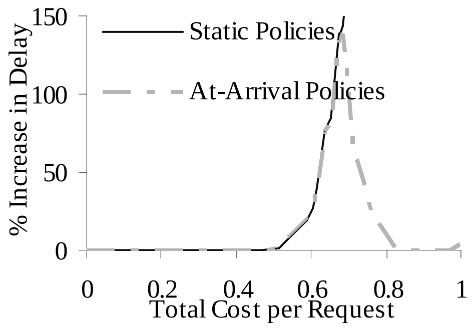
where $\partial_{k1=0} = 1$ if $k_1 = 0$, and 0 otherwise. Using the above threshold values the total service delivery cost C can be calculated for any given configuration.

The best performance of the policy is obtained using the best possible system configuration. Assuming that all arrival rates are known and a target delay A^* is selected, the total service delivery cost C, under optimal parameters, can therefore be calculated by taking the minimum over all possible configurations (i.e., possible choices of k_1 and k_2),

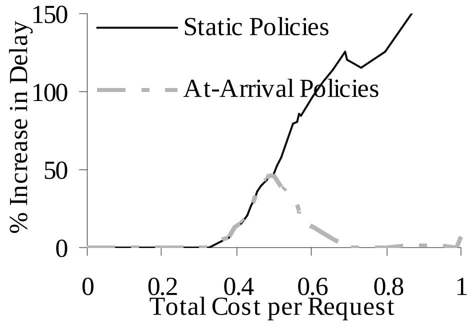
$$\min_{k_1,k_2}\left\{\frac{L\left(\lambda_1 + \sum_{i=k_2+1}^{N}\lambda_i\right)}{n_1 + f_1} + \sum_{i=2}^{k_1}\frac{L\lambda_i}{n_1 + f_1} + \frac{L\sum_{i=k_1+1}^{k_2}\lambda_i}{n_{k_1+1} + f_{k_1+1}} + \right.$$
$$\left. + c\left[\frac{\sum_{i=k_2+1}^{N}\lambda_i}{\sum_{i=1}^{N}\lambda_i} + \frac{\sum_{i=k_1+1}^{k_2}\lambda_i}{\sum_{i=1}^{N}\lambda_i}\left(\frac{n_g + f_g - 1}{n_g + f_g}\right)\left(1 - \sum_{i=k_1+1}^{k_2}\left(\frac{\lambda_i}{\sum_{j=k_1+1}^{k_2}\lambda_j}\right)^2\right)\right]\sum_{i=1}^{N}\lambda_i L\right\}. \tag{4.14}$$



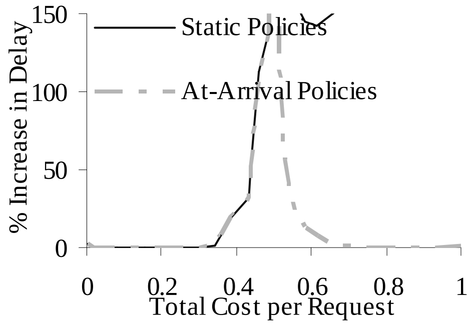
(a) $c = 0.3$, $N = 16$, equal request rates



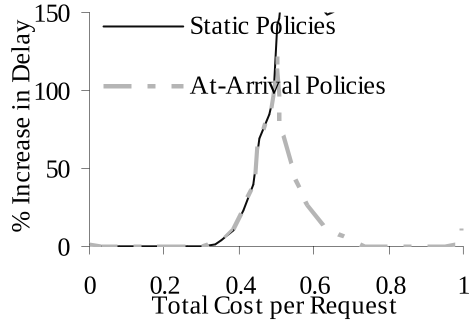
(b) $c = 0.7$, $N = 16$, equal request rates



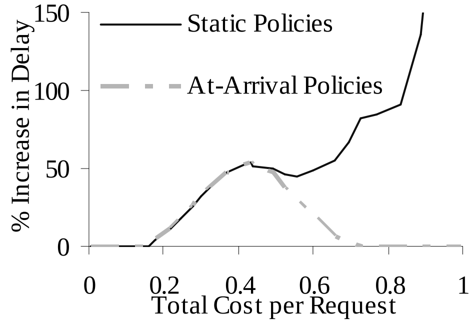
(c) $c = 0.5$, $N = 4$, equal request rates



(d) $c = 0.5$, $N = 64$, equal request rates



(e) $c = 0.5$, $N = 16$, equal request rates



(f) $c = 0.5$, $N = 16$, $\lambda_i = \Omega/i$

Figure 4.10: Best Potential Performance with Static Policies and Actual Performance with an At-arrival Policy, Relative to the Actual Performance for a Global Dynamic Policy, for Batched Service (each policy is evaluated based on average delay).

4.5.4 Performance Comparisons

Figure 4.10 compares the performance of optimal static policy and the candidate at-arrival policy, both evaluated analytically, to the performance of the dynamic on-line policy, evaluated using simulations. As shown, the performance differences using an average delay metric is similar to the corresponding differences using maximum delay as the metric (Figure 4.9). However, while it is clear that there is a large benefit from using dynamic policies, due to higher complexity and no delimiting at-arrival policy, the results using average delay as a metric is less clear. Based on the results obtained, it appears beneficial to defer replica selection decisions as late as possible. With the higher complexity of these policies, no local state policies have been considered. The design and evaluation of such policies remain an open problem.

4.6 Summary

To summarize, this chapter considers the fundamental conflict between replication and service aggregation. Specifically, to determine an appropriate level of complexity, this section compares policy classes of varying complexity. Policy classes are defined and evaluated with respect to both maximum and average client delay, under both a batched and a fountain service model. It is concluded that (i) using dynamic state information (rather than only proximities and average loads) can yield large improvements in performance, (ii) when it is possible to defer selection decisions (e.g., when requests are delayed and served in batches), deferring decisions as late as possible, rather than using at-arrival selection, can yield significant improvements, although only for a fairly narrow range of the model parameter space, and (iii) relatively simple maximum delay policies using "local state" information appear able to achieve most of the potential performance gains (achieved by "global state" policies).

Chapter 5
On-demand Streaming using Scalable Download

Existing peer-assisted systems and algorithms proposed for on-demand streaming of stored media files [24, 53, 161] establish relatively long-duration streams from the content source and between peers as organized into some form of overlay topology. Rather than requiring peers to organize themselves into such structures, this chapter proposes using adaptations of already proposed peer-assisted scalable downloading techniques, such as BitTorrent [48], to achieve a form of "streaming" delivery, in the sense that playback can begin well before the entire media file is received. With BitTorrent-like protocols, the file is split into smaller pieces that each peer can download in parallel, from multiple peers. A peer can download pieces from *any* other peer (that has at least one piece that the peer does not have itself). To encourage peers to contribute with upload bandwidth, each peer prefers to upload to peers that upload to it, at a relatively high rate. This approach is very flexible and its simplicity allows the system to easily handle dynamic environments where peers may join and/or leave the system frequently.

In the context of file *download*, where the file is not considered usable until fully downloaded, it has been found beneficial to download pieces in an order that maintains high piece diversity [110, 111]. For example, BitTorrent uses a *rarest-first* policy when deciding which pieces to download. With this policy, strict preference is given to pieces that are the rarest among the set of pieces owned by all the peers from whom it is downloading. This ensures that peers are more likely to have pieces to share. On the other hand, in the context of *streaming*, where clients may start playback before the content is fully retrieved, it is most natural to download pieces *in-order*. When designing piece selection techniques for this context it is therefore important to achieve a good compromise between the goal of piece diversity and in-order retrieval of pieces. Furthermore, an on-line policy is needed for deciding when playback can safely commence. This chapter presents and evaluates both piece selection policies and simple startup rules.

Various policies are evaluated using event-based simulations, in which each peer is assumed to be bottlenecked by either its maximum sustainable client transmission or reception rate (called the peers upload and download bandwidth capacity, respectively). It is further (very conservatively) assumed that no peer, except the original content source, shares pieces once the whole file has been received. In a real system peers are likely to continue serving other peers as long as they are still playing out the media file, while other peers may (graciously) choose to upload to other peers beyond that time. With the higher availability of rare pieces and additional seed bandwidth in such systems, the benefits of more aggressive piece selection techniques (giving priority to earlier pieces rather than rare pieces) are likely to be even greater than presented here.

Simple probabilistic piece selection techniques are proposed that achieve a good tradeoff between selecting pieces that give priority to pieces needed sooner, while maintaining enough piece diversity that peers can easily exchange pieces among each other. These techniques are shown to enable startup delays significantly smaller than the download time. Secondly, a number of simple policies to determine when to safely begin playback are evaluated. A simple rule is found promising, which requires the number of pieces retrieved to exceed some (small) threshold, and the rate at which in-order pieces are retrieved to exceed a threshold that would allow playback without interruptions, should that rate be maintained.

The remainder of the chapter is organized as follows. The simulation model is described in Section 5.1. Section 5.2 defines a number of candidate piece selection policies and evaluates their potential performance advantages. Section 5.3 addresses the problem of dynamically determining the startup delay. Conclusions are presented in Section 5.4.

5.1 Simulation Model

In contrast to previous simulation studies [25, 78] that only allow peers to be connected to a few peers, this study assumes that peers are connected to all other peers in the system. In real systems peers are often connected to many peers; for example, the default parameters in recent versions of the mainline client allow peers to be connected to up to 80 other peers, which is often achieved in practice [111]. It is further assumed that pieces are split into infinitesimal sub-pieces, such that any portion of a piece can be uploaded to, or downloaded from, a peer.[31]

The model assumes that a peer *i* can at most have v_i concurrent upload connections and no connections are choked in the middle of an upload. The set of peers that it is uploading to changes when (i) it completes the upload of a piece, *or* (ii) some other peer becomes interested and peer *i* does not utilize all its upload connections. The new set of upload connections consists of (i) any peer currently in the middle of an upload, *and* (ii) additional peers up to the maximum limit v_i. These additional peers are determined according to a rate-based unchoking algorithm. To model rate-based tit-for-tat with optimistic unchoking, a probabilistic approach is used. With a probability $1/n_i$ the next peer to get unchoked is selected using an optimistic unchoking policy, and with a probability of $(v_i–1)/v_i$ using a rate-based policy. The optimistic unchoking policy selects a random peer from the set of peers that are interested. The rate-based policy selects the peer, from within the set of interested peers, which is uploading to peer *i* at the highest rate. Random selection is used to break ties. Note that this ensures that the seeders only use random peer selection.

To simulate the exchange of pieces among peers it is important to determine the rate at which data is exchanged. Connection bottlenecks are assumed to be located at the end points; i.e., connections are either bottlenecked by the upload bandwidth capacity (i.e., the maximum sustainable client transmission rate) at the sender or by the download bandwidth capacity (i.e., the maximum sustainable client reception rate) at the receiver. It is further assumed that the network operates using max-min fair bandwidth

[31] The size of the sub-pieces used in BitTorrent is typically 1/16 of the size of a piece (16 kB versus 256 kB).

Table 5.1: Notation used in Chapter 5.

Symbol	Definition
N	Number of peers
u_i	Upload bandwidth capacity of peer *i*
b_j	Download bandwidth capacity of peer *j*
u_i^m	Maximum sustainable upload rate for any of *i*'s upload connections
b_j^m	Maximum sustainable download rate for any of *j*'s download connections
δ_{ij}	1 if *i* is transmitting to *j* (i.e., *j* is *interested* in *i* and has been *unchoked* by *i*); 0 otherwise
r_{ij}	Connection rate from peer *i* to peer *j*
λ	File request rate
φ	Peer defection rate
η	Exponential decay factor

sharing (using TCP, for example) [23, 99, 136, 183]. In max-min fair networks each flow operates at the highest possible rate that ensures that (i) no bottleneck operates above its capacity, and (ii) the rate of no flow can be increased without decreasing the rate of some other flow operating at a the same or lower rate (without exceeding some bottleneck's maximum sustainable bandwidth capacity).

A bottleneck is constrained whenever the total flow through the bottleneck is equal to its capacity and a flow is considered constrained whenever one of its bottlenecks is constrained. Existing algorithms [23] to determine the max-min fair bandwidth allocation, iteratively identify the next bottleneck that would become constrained if the rate of all unconstrained flows in the network was increased by the same amount. At the point a bottleneck becomes constrained, all flows passing through the bottleneck (that are not yet constrained) become constrained and their rates can no longer be increased. With more constrained flows the rates of the remaining unconstrained flows can again be increased uniformly, until another bottleneck becomes constrained. This procedure is repeated until all flows are constrained. To reduce the computation cost, the algorithm is modified to take into consideration that each end-to-end connection (or flow) is either constrained by its upload or download connection. Rather than identifying one bottleneck at a time this symmetry is used to determine (potentially) multiple bottlenecks (with different allowable rates) in the same iteration.

Using the notation defined in Table 5.1, Figure 5.1 presents the algorithm used to find the max-min fair solution. Given a set of unchoked connections (defined through

```
$S = \{u_i^m\} \cup \{b_j^m\}$
$while \quad |S| > 0$
    $(\forall i \mid u_i^m \in S): \quad u_i^m = \left( u_i - \sum_{\{j \mid b_j^m \notin S\}} \delta_{ij} b_j^m \right) \Big/ \sum_{\{j \mid b_j^m \in S\}} \delta_{ij}$
    $(\forall j \mid b_j^m \in S): \quad b_j^m = \left( b_j - \sum_{\{i \mid u_i^m \notin S\}} \delta_{ij} u_i^m \right) \Big/ \sum_{\{i \mid u_i^m \in S\}} \delta_{ij}$
    $u^* = \min_{\{i \mid u_i^m \in S\}} u_i^m; \quad b^* = \min_{\{j \mid b_j^m \in S\}} b_j^m$
    $if \; u^* < b^*, then \quad S = S \setminus \{u_i^m < b^*\}$
    $else \; if \; b^* < u^*, then \quad S = S \setminus \{b_j^m < u^*\}$
    $else \quad S = S \setminus (\{u_i^m \le u^*\} \cup \{b_j^m \le b^*\})$
$end \; while$
$\forall i \forall j : r_{ij} = \delta_{ij} \min\{u_i^m, b_j^m\}$
```

Figure 5.1: Max-min Fair Bandwidth Allocation Algorithm used to Calculate the Rates of the Unchoked Peer Connections.

δ_{ij}), as well as the total uplink capacity u_i and downlink capacity b_j for each peer ($1 \le i,j \le N$) the algorithm determines the maximum upload rate u_i^m among all upload connections from peer i and the maximum download rate b_j^m among all download connections for peer j. The algorithm starts with the set of maximum upload and download rates (u_i^m and b_j^m) undetermined (within the set S). As long as at least one of these values is undetermined, the algorithm calculates a lower bound estimate of the undetermined values by providing equal share of the remaining bottleneck capacity, with the rate of the already constrained flows subtracted. Since these values can only increase in later iterations, u^* and b^* provide lower bound estimates of the smallest upload and download constraints of future iterations, which itself implies that (i) all flows with lower upload rate u_i^m than the most constrained downlink b^* will be upload constrained by u_i^m, and (ii) all flows with lower download rate b_j^m than the most constrained uplink u^* will be download constrained by b_j^m. This observation allows (potentially) multiple u_i^m or b_j^m values to be determined (and removed from the set of still unconstrained variables S) in each iteration. With each flow only having two bottlenecks, each characterized by u_i^m or b_j^m, the rate of the flow between peer i and j, denoted r_{ij}, can easily be calculated as $\delta_{ij} \min\{u_i^m, b_j^m\}$.

Table 5.2: Example Simulation Execution Times for a Flash Crowd of Size *N*

N	**Execution Times (in seconds)**
16	1.6
32	5.6
64	24.8
128	179
256	1,560
512	8,290

Assuming a total of N fully connected peers, each iteration requires $O(N^2)$ operations, one for each (possible) connection. In the worst case, this algorithm requires $2N$ iterations, resulting in a total of $O(N^3)$ operations. However, typically this algorithm requires many fewer iterations. For example, if all connections are upload constrained, the algorithm allows the rates of all flows to be determined using only two iterations, while the standard algorithms would require N iterations (one for each uplink bottleneck, until the rates of all flows are determined). Also, in systems where peers are not connected to all other peers the number of calculations per iteration can easily be reduced (e.g., using sparse matrices).

Using event-based simulations, rates must be recalculated each time the set of unchoked peers changes for some peer (i.e., with a large number of peers, roughly each instance at which a piece becomes fully downloaded). Therefore, with a large number of pieces, and given the cost of the above algorithm, the computational cost of these simulations is restricting simulations to only smaller peer populations. Table 5.2 illustrates some example execution times on an AMD Opteron 850 processor running at 2.4GHz, for a scenario with a flash crowd of N peers, each downloading all 512 pieces of a file using a rarest-first policy.

5.2 Piece Selection

This section considers the order in which pieces should be retrieved to allow streaming. A peer requests a new piece each time it gets unchoked by a peer in which it is interested, or when the download of one piece is completed and the peer is still interested in additional pieces. Section 5.2.1 describes candidate piece selection policies, and Section 5.2.2 evaluates these policies with regards to the best possible startup delay (i.e., the startup delay peers would be able to achieve if knowing when each piece would be fully retrieved). In practice this exact time instance can not be determined using an on-line algorithm. Section 5.3 evaluates simple on-line policies to determine when to start playback.

5.2.1 Candidate Policies

To begin playback well before the entire media file is retrieved, pieces must be selected in a way that effectively mediates the conflict between the goals of high piece diversity (achieved in BitTorrent using a *rarest-first* policy) and the *in-order* requirements of media file playback. Assuming that peer j is about to request a piece from peer i, two baseline policies are defined as follows:

Rarest: Among the set of pieces that peer i has and j does not have, client j requests the rarest piece among the set of all pieces that peers that j is connected to have. Ties are broken using random selection.

In-order: Among the set of pieces that peer i has, client j requests the first piece that it does not have itself.

Rather than considering advanced piece selection techniques, simple probabilistic policies are proposed. Perhaps the simplest such technique is to request an in order piece with some probability and the rarest piece otherwise. Other techniques may use some probability distribution to bias towards earlier pieces. The Zipf distribution has been found to work well for this purpose. Below, one of each of these two types of probabilistic policies are defined:

Portion (*p*): For each new piece request, client j uses the in-order policy with a probability p and the rarest policy with a probability $(1–p)$.

Zipf (α): For each new piece request, client j probabilistically selects a piece from the set of pieces that i has, but that j does not have. The probability of selecting each of these pieces is chosen to be proportional to $1/(k+1–k_0)^{\alpha}$, where k is the index of the piece, k_0 the index of its first missing piece, and α is a parameter of the protocol.

The policies choices considered in this section are: rarest, in-order, portion(50%), portion(90%), and Zipf(1.25). With the portion policy, the most natural choice is to use a probability p of 50%. However, in some scenarios this parameter choice is very conservative. To illustrate the performance of a more aggressive parameter choice, simulations are also presented in which in-order pieces are selected with a probability of 90%. For the Zipf policy, a Zipf parameter α of 1 may be the most natural choice; however, a slightly more aggressive parameter choice (in this case 1.25) is generally beneficial. Results using other parameter choices for the Zipf policy are not presented as Zipf(1.25) performs well in all scenarios considered. However, note that the Zipf parameter can be tuned so that the policy is more or less aggressive. For example, with a larger Zipf parameter the policy becomes more aggressive, giving more bias to earlier pieces, while with a smaller Zipf parameter it becomes less aggressive.

5.2.2 Performance Comparisons

Throughout this chapter, it is assumed that there is one single persistent seed and all other peers leave the system as soon as they have retrieved the entire file (i.e., only acts as leechers). As noted previously, this is a very conservative assumption, as in real systems peers are likely to continue serving other peers as long as they are still playing out the stream, while other peers may (graciously) choose to serve as seeds beyond that.

Without loss of generality, the file's size L and play rate r_p are both set at 1. This corresponds to measuring the volume of data transferred in units of the file size and time in units of the file play duration. Hence, all rates are expressed relative to the play rate, and all startup delays are expressed relative to the time it takes to play the entire file. For example, an achieved download rate of 2 means that the file can be downloaded approximately twice as fast as it can be played out. Similarly, a startup delay of 0.1 means that the client could start playback after a duration equal to 0.1 times the play duration. The file is split into 512 pieces, and unless stated otherwise, peers are assumed to have three times higher download capacity than upload capacity and each peer uploads to at most four peers simultaneously.

To compare the performance of the above piece selection policies, this section initially considers a simple scenario in which peers (i) do not leave the system until having fully downloaded the file (i.e., the peer defection rate φ is equal to zero), (ii) arrive according to a Poisson process, and (iii) are homogenous (i.e., all peers have the same upload bandwidth capacity u and download bandwidth capacity b). Alternative scenarios and workload assumptions are subsequently considered.

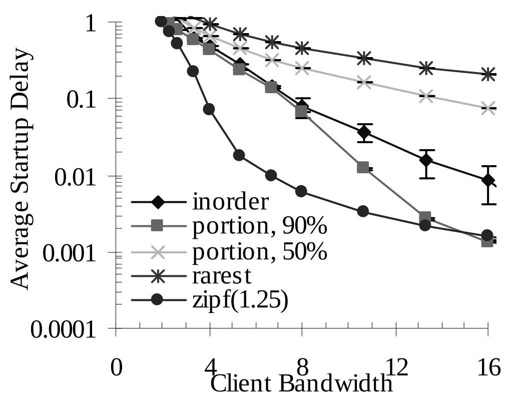


Figure 5.2: Average Achievable Startup Delay under a Steady State Poisson Arrival Process without Early Departures: The Impact of Client Bandwidth Capacity (b/u = 3, λ = 64, and φ = 0).

To capture the steady state behavior of the system, the system is simulated for at least 4000 requests. Further, measurements are only done for requests which do not occur in the beginning or the end of each simulation. Typically, the first 1000 and the last 200 requests are not analyzed; however, to better capture the steady state behavior of the inorder policy warmup period longer than 1000 requests is sometimes required.[32] Each data point represents the average of 10 simulations. Unless stated otherwise, this methodology is used throughout Chapter 5. It should be noted that the accuracy in these values are high. To illustrate this, Figure 5.2 shows confidence intervals capturing the true average with a confidence of 95%. Note that the confidence intervals are only visible for the inorder policy. For the other policies the average values presented in this chapter are very accurate and confidence intervals are therefore omitted.

Figure 5.2 characterizes the system (under this scenario) by varying the total client bandwidth capacity (i.e., $u + b$). The peer arrival rate λ is assumed to be 64 and the seed has an upload bandwidth capacity equal to that of regular peers. The most significant observation is that Zipf(1.25) consistently outperforms the other candidate policies. In systems with an upload bandwidth capacity at least twice the play rate (i.e., $u \geq 2$) peers are able to achieve startup delays two orders of magnitude smaller than the time it takes to play the file and much faster than it would take to download the file using the rarest policy.

[32] The inorder policy was typically simulated using at least 20,000 requests.

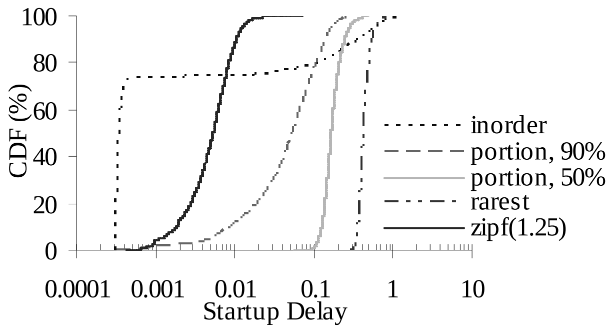
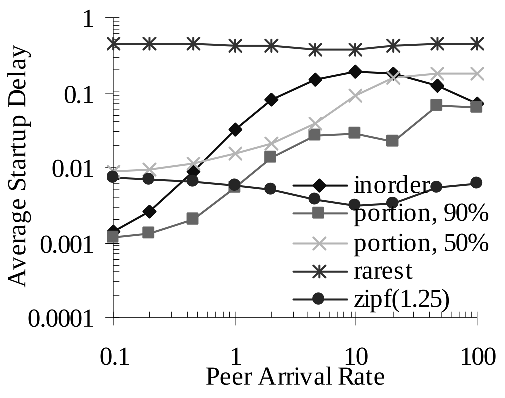
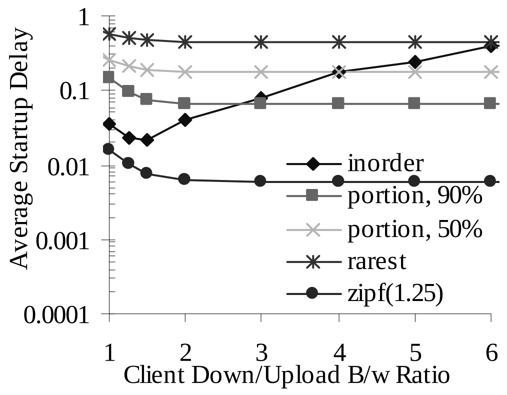


Figure 5.3: Cumulative Distribution Function of the Best Achievable Startup Delay under a Steady State Poisson Arrival Process without Early Departures (u = 2, b = 6, λ = 64, and φ = 0).

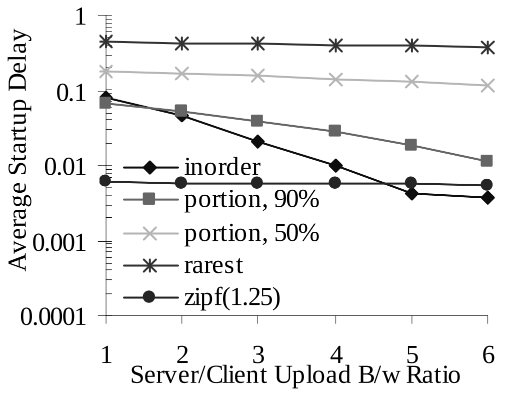
Figure 5.3 presents the cumulative distribution of achievable startup delays under this initial scenario. Note that Zipf(1.25) achieves low and relatively uniform startup delays for the set of clients. The high variability in startup delays, using the in-order policy, are due to peers becoming synchronized. Peers using the in-order policy never upload pieces to peers they currently are downloading from (as those peers have all pieces that the peer has). Therefore, all active upload connections are determined using random unchoking. With larger download capacity and many peers to download from, peers with fewer pieces will quickly catch up with peers who have fewer peers to download pieces from. This causes peers to become synchronized, all requiring the same piece. Being limited by the upload rate of the seed these peers will, at this point, see poor download rates. In general, some peers will be stuck in such “queue build-up” for a long time, while others will be stuck for a much shorter time (possibly still allowing them low startup delays). With lots of peers completing their downloads at roughly the same time, the system will become close to empty, before a new set of peers repeat this process (in which they all become synchronized). This service behavior causes the number of peers in the system using the in-order policy to follow a saw-tooth pattern. In contrast, the number of concurrent leechers, using the other policies, is relatively stable.



(a) The Impact of the Peer Arrival Rate λ (u = 2, b = 6, η = 0, φ = 0)



(b) The Impact of the Ratio Between the Clients Download and Upload Bandwidth Capacity b/u (u = 2, λ = 64, φ = 0)



(c) The Impact of the Bandwidth Capacity of the Persistent Seed (u = 2, b = 6, λ = 64, φ = 0)

Figure 5.4: Average Achievable Startup Delay under Steady State Poisson Arrival Process: Example Scenarios.

Figure 5.4 considers the impact of (i) the peer arrival rate, (ii) the ratio between peers' download and upload bandwidth capacity, and (iii) the bandwidth capacity of the persistent seed. Again, the simple Poisson model without early departures and homogenous clients is considered. As expected, in-order and portion(90%) do very well in systems with very low arrival rates. However, already at an arrival rate of one Zipf(1.25) outperforms these policies. In fact, Zipf(1.25) is relatively insensitive to the peer arrival rate. At this point it should be noted that the decrease in average delay, observed by the in-order policy, may be somewhat misleading as the achievable startup delays in this region is highly variable (see Fig 5.3). Figure 5.4(b) illustrates that the results are relatively insensitive to the download/upload bandwidth-capacity ratio for ratios larger than 2. In this experiment the upload bandwidth capacity *u* is fixed at 2 and the download bandwidth capacity *b* varied. Typical Internet connections generally have ratios between 2 and 8 (e.g., [156]). While the in-order policy achieves improved performance when the ratio is constrained between 1 and 2, after that its performance decreases. The reason for these increasing startup delays is that recently arrived peers, that have many peers from which they can download, will get a larger share of the seeds total upload bandwidth capacity, than in systems where these peers are (more) download constrained. As pieces uploaded to such peers are of no benefit to peers waiting for later pieces, and the retrieved piece could have been retrieved from any of these peers, the seed bandwidth used to deliver such a piece is essentially wasted.

Finally, this chapter considers a scenario in which the persistent seed may be more powerful than regular peers. For simplicity it is assumed that the seed (or server) is behind a single bottleneck and that it allows for the maximum number of upload connections to be proportional to the capacity of the seed. A server with twice the upload bandwidth capacity of a regular peer can therefore upload to twice as many peers in parallel as a regular peer (assuming enough peers are interested), or upload to a single peer at twice the maximum sustainable upload rate of a peer (assuming only one peer is downloading). Figure 5.4(c) illustrates that (for systems in steady state) not much additional server bandwidth capacity is required for peers to achieve low average startup delays using much more aggressive policies (such as in-order). Similar improvements can be achieved using a more aggressive Zipf parameter.

This section now considers three additional scenarios, called the second, third and fourth scenario. The second scenario considers the average performance of peers in a system in which some peers leave before having fully downloaded the file. It is assumed that peers arrive according to a Poisson process but may leave the system prematurely at a fixed rate of φ (per client). Figure 5.5 illustrates that all policies are insensitive to the rate peers depart the system. This insensitivity to peer departures is a characteristic of peers not relying on retrieving pieces from any particular peer. This insensitivity has been verified by reproducing very similar graphs to those presented in Figures 5.2, 5.3 and 5.4.

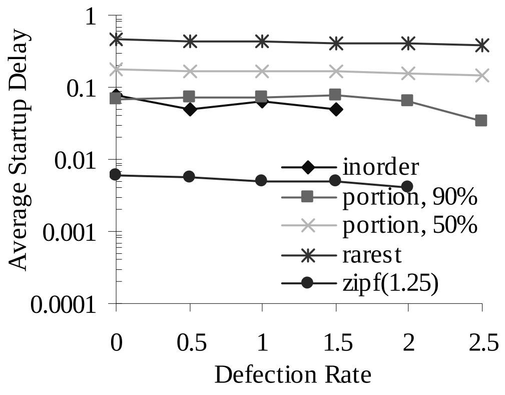


Figure 5.5: Average Achievable Startup Delay under a Steady State Poisson Arrival Process with Early Departures (u = 2, b = 6, λ = 64).

The third scenario considers the impact of peer arrival patterns. Here, peers are assumed to arrive according to a Poisson process with exponentially decaying arrival rate. This type of arrival pattern is motivated by measurement studies done on real BitTorrent systems [82]. Given an initial arrival rate λ_0 and an exponential decay factor η the rate at a time instance t can be calculated as $\lambda(t) = \lambda_0 e^{-\eta t}$. Using this arrival pattern, $100 \times (1 - e^{-\eta t})$ percent of the total number of arrivals occurs within time t. By varying the decay factor between 0 and ∞ both a pure Poisson arrival process (in steady state) and a flash crowd in which all peers arrive instantaneously (before and after which no other arrivals occur) can be captured. To compare arrival patterns with different decay factors the expected number of arrivals within some time period is fixed at some value. In Figure 5.6 the expected number of arrivals within the first 2 time units (i.e., the time it takes to play the file twice) is set to 128, and the exponential decay factor γ is varied between 0.01 and 100. To put this range of decay factors into perspective, with a decay factor η = 1, 63.2% of all peer arrivals occur within the first time unit and 86.5% within the first two time units. With a decay factor η = 6.9, on average 99.9% of the peer arrivals occur within the first time unit. For these experiments no warmup period were used and simulations were run until the system were empty. Again, note that the performance of in-order and portion(90%) quickly becomes very poor, as the initial arrival rate λ_0 increases. These policies do a poor job ensuring that peers have pieces to

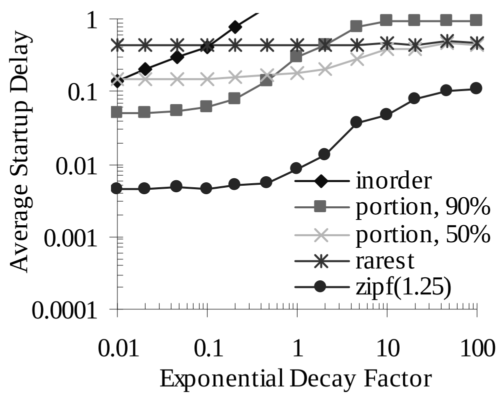


Figure 5.6: Average Achievable Startup Delay under Exponentially Decaying Arrival Rates (u = 2, b = 6, $\lambda(t) = \lambda_0 e^{-\eta t}$, $\lambda_0 = 128\eta / (1 - e^{-2\eta})$).

share among each other, and are therefore limited by the upload bandwidth capacity of the seed.

The fourth scenario considers a system in steady state, but with two classes of peers: low and high-bandwidth peers. The low-bandwidth peers have a total client bandwidth capacity of 1.6 (u_L = 0.4 and b_L = 1.2) and high-bandwidth clients have a total client bandwidth capacity of 8 (u_H = 2 and b_H = 6). Figure 5.5 illustrates the average startup delay for the high-bandwidth peers as a function of the percentage of low-bandwidth peers arriving to the system. As expected, a large portion of low-rate peers causes significant performance degradation to high-rate peers. The figure for low-bandwidth peers looks very similar, but with the exception that the minimum startup delay for the policies is higher (e.g., the minimum startup delay using Zipf(1.25) is roughly 0.08).

Similar results have also been observed in a scenario where all peers are assumed to have a total upload bandwidth capacity of 8 (u = 2 and b = 6); however, one group only makes 20% of its upload bandwidth capacity available (i.e., u_L = 0.4 and u_H = 2). For this modified scenario the startup delays of Zipf(1.25) improve slightly as the low sharing peers provide additional download capacity to the system. With the in-order policy, on the other hand, the startup delays become much worse (as the number of low-rate peers increase). With these peers downloading more data from the seed (allowed by its higher download bandwidth capacity) more seed bandwidth is used uploading to weaker peers, which do not do a good job relaying these pieces to other

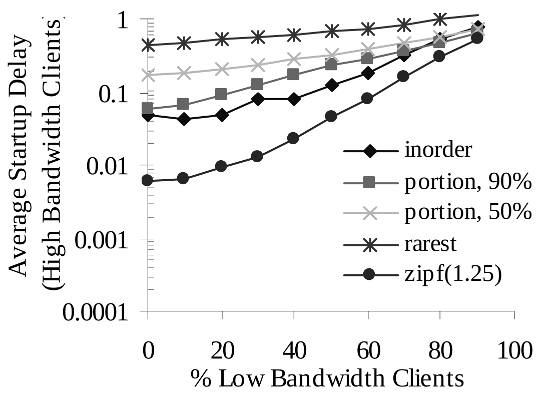


Figure 5.7: Average Achievable Startup Delay under a Steady State Poisson Arrival Process with both High and Low Bandwidth Clients ($\lambda = 64$, $\varphi = 0$, $u_L = 0.4$, $b_L = 1.2$, $u_H = 2$, $b_H = 6$).

peers. It should also be noted that, in systems using Zipf(1.25), low sharing clients may, on average, see startup delays of more than twice those of regular clients. This discrimination is especially pronounced in the region where the clients start to see degrading performance.

5.3 Using a Dynamic Startup Rule

In highly unpredictable environments, with large and changing sets of peers, it is difficult to predict future download conditions. Therefore, it is not expected that any on-line strategy for selecting a startup delay would give close to optimal startup delays (without significant chance of playback interruption). To deal with (potentially) missing pieces, it is likely that existing techniques used by existing media players (such as error concealment, layered media, etc.) may be applied. This section present a simple protocol that uses the Zipf(1.25) policy, presented in the previous section, together with a simple policy to predict when playback can safely commence. Maintaining the simplicity of the piece selection policy, it should be noted that only the startup policy requires future rates to be predicted. Section 5.3.1 defines a number of candidate policies, while their performance is compared in Section 5.3.2.

5.3.1 Simple Startup Policies

This section defines three simple startup policies. The simplest policy is to always ensure that at least some minimum number of pieces is retrieved before allowing playout to begin. Here, such a policy is defined as follows:

At-least (k_{req}): Start playback when at least k_{req} pieces have been retrieved, and one of those pieces is the first piece of the file.

While this policy does not require the download rates to be predicted, determining the best possible value of k_{req} is non-trivial (in highly dynamic environments).

The other two policies considered here attempt to measure the rate at which a file marker, before which all the file data has been retrieved, advances through the file. An "in-order buffer" is defined that contains all pieces up to the first missing piece, and the rate at which the size of this buffer increases is denoted by r_{seq}. Note that r_{seq} will initially be smaller than the download rate (as some pieces are retrieved out-of-order), but can exceed the download rate as holes (of missing pieces) are filled. Assuming a constant valued download rate, r_{seq} can be expected to increase over time, as holes are filled more and more frequently. Assuming a constant download rate, therefore, it is safe to start playback as soon as r_{seq} allows the in-order buffer to be filled within the time it takes to play the entire file. With k pieces in the in-order buffer r_{seq} must therefore be at least $(K–k) / K$ times as large as the play rate, where K is the total number of pieces in the file. Using this rate condition two rate-based policies are defined:

LTA (k_{req}): Start playback when the start condition of at-least(k_{req}) is satisfied *and* the rate condition is satisfied by $r_{seq} = (Lk/K)/T$, where T is the time since the peer arrived to the system and k is the index of the piece immediately before the first missing piece.

EWMA (k_{req}, α): Start playback when the start condition of at-least(k_{req}) is satisfied *and* the rate condition is satisfied by $r_{seq} = (L/K) / \tau_{seq}$, where τ_{seq} is calculated using an exponentially weighted moving average (EWMA).

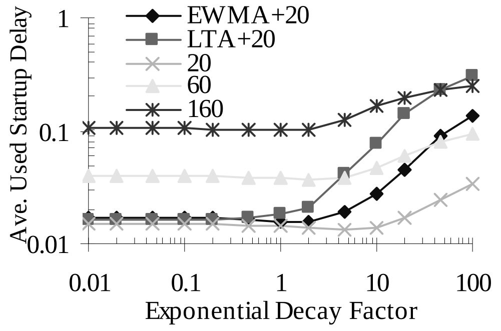
For all results presented here, the EMWA uses an exponential weight factor equal to 0.1 (i.e., the old estimation of τ_{seq} is given a weight 0.9 and the observed inter-arrival time is given a weight 0.1). τ_{seq} is initialized at the time the first piece is retrieved (using its inter-arrival time). When multiple pieces are inserted into the in-order buffer at the same piece arrival, equal weight is given to each piece. For example, if piece 3 was retrieved *t* before piece 4. Assuming piece 5 and 6 have already been retrieved at the time piece 4 is retrieved, each of the three requests are considered as if retrieved in-order with inter-arrival time *t*/3.

5.3.2 Performance Comparisons

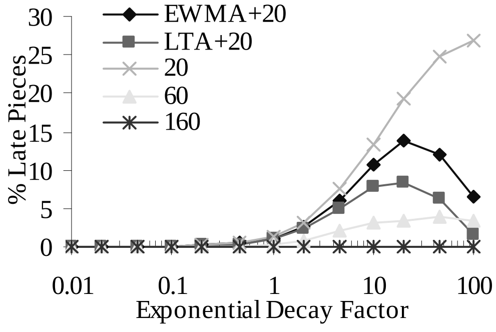
Making the same workload assumptions and using the same simulation methodology as in Section 5.2.2, the above startup policies are evaluated together with the Zipf(1.25) piece selection policy. Whereas startup policies may be tuned for the conditions under which they are expected to operate, in highly dynamic peer environments with changing network conditions, it is important for such policies to be responsive, allowing the startup delays used to adapt as the network condition changes.

To evaluate the above policies over a wide range of network conditions, scenario three and four from the previous section is used, together with their corresponding simulation setup. In the scenario three the burstiness with which peers arrive is varied. Here, the exponential decay factor is varied four orders of magnitude, covering arrival patterns from close to steady state to a flash crowd (in which case all peers arrive close to instantaneously). In scenario four arriving peers belongs to one of two classes, high- and low-bandwidth clients. For this scenario the portion of peers is varied such that the network conditions is varied from good (where most peers are high-bandwidth clients) to a case with poor network conditions (where the majority of peers are low-bandwidth clients). For both these scenarios, the following policies are compared: at-least(20), at-least(60), at-least(160), LTA(20), and EWMA(20, 0.1).

Figures 5.8 and 5.9 presents the average used startup delay and the percentage of pieces not retrieved in time of playback of that part of the file. Again, note that missing pieces could be handled using various existing techniques, designed to handle missing data (such as error concealment, layered media, etc.). Figure 5.8 presents the results for varying arrival patters. These results suggests that both LTA(20) and EWMA(20,0.1) adjust well to the changing network conditions. For close to steady arrival rate they both achieve low startup delays and as conditions become burstier, adjust their startup delays to maintain a low percentage of late pieces. Of these two policies, LTA is



(a) The Average used Startup Delay

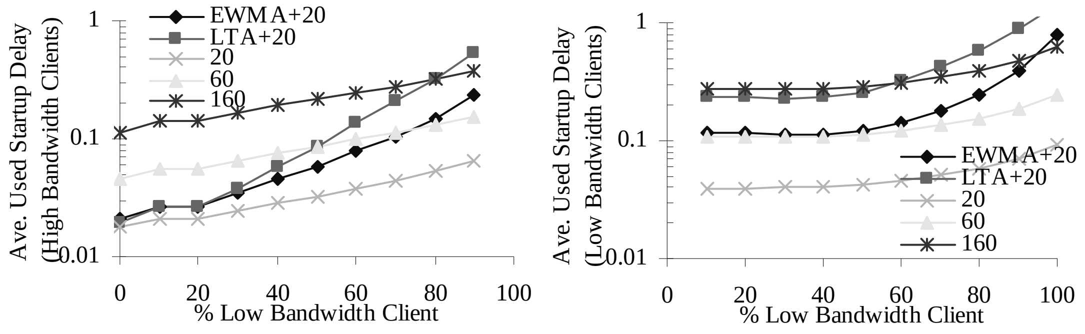


(b) The Percentage Late Pieces

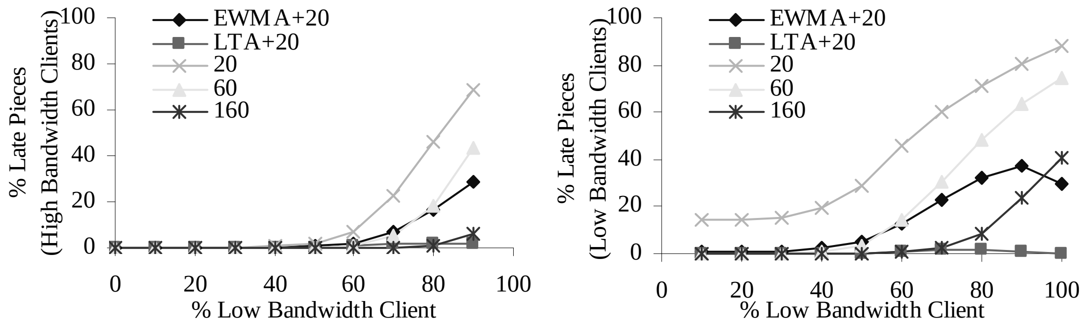
Figure 5.8: Exponentially Decaying Arrival Rates (u = 2, b = 6, $\lambda(t) = \lambda_0 e^{-\eta t}$, $\lambda_0 = 128\eta / (1 - e^{-2\eta})$).

somewhat more conservative and is therefore able to achieve lower loss rates. While the *b*-parameter can be tuned for certain network condition (e.g., in the upper part of the parameter space at-least(60) achieves both lower delays and lower percentage of late pieces than EWMA and LTA), it is not necessarily good in practice, as network conditions may quickly change.

Figure 5.9 illustrates similar graphs for high- and low-bandwidth clients as a function of the percentage of arriving clients that are low-bandwidth clients; the differences in responsiveness are even greater for this scenario. By increasing the startup delays LTA(20) effectively adapts its startup delays such that the percentage of late pieces is consistently low and the delays used are comparable with all other policies achieving low loss rates for a particular peer mix. EWMA(20) on the other hand is somewhat more aggressive, resulting in larger percentages of late pieces, whereas the at-least(b) policies are non-responsive. This is best illustrated by the straight lines and/or high loss rates observed by this policy. Designed for highly dynamic environments, the characteristics observed by the LTA(b) policy is found promising. This policy is relatively simple and uses a single parameter, in contrast to EWMA(b,α) which requires two parameters. Using a long term average (LTA) of the rate at which the size of the in-order buffer have changed, rather than an exponentially weighted moving average (EWMA), giving more bias towards more recently retrieved pieces, this policy is somewhat more conservative allowing the policy to avoid being fooled by temporary increases in the rate at which this buffer changes.



(a, b) The Average Startup Delay used by High and Low Bandwidth Clients, Respectively.



(c, d) The Percent Pieces that are not Retrieved in Time of Playback, for High and Low Bandwidth Clients, Respectively.

Figure 5.9: Heterogeneous Scenario with Poisson Arrivals with both High and Low Bandwidth Clients (λ = 64, φ = 0, u_L = 0.4, b_L = 1.2, u_H = 2, b_H = 6).

5.4 Summary

This chapter proposes a new approach to achieve peer-assisted streaming, in which adaptations of already existing peer-assisted download protocols, such as BitTorrent, are used to download pieces of a file in an order that allows streaming. Simple probabilistic piece selection policies are shown to allow peers to begin playback well before the entire file is downloaded. While giving preference to earlier pieces, these policies effectively mediate the conflict between the goals of achieving high piece diversity (required for peers to effectively share pieces) and the in-order requirements of media file playback. Further, promising results are obtained using a simple rule for when to start playback, which requires the number of pieces to exceed some (small) threshold, and the rate at which in-order pieces are retrieved to allow playback to begin (if that rate was to be maintained).

Chapter 6
Conclusions

This thesis is concluded with a short summary, a list of the main contributions, as well as a brief outline of potential future directions.

6.1 Thesis Summary

Chapters 1 and 2 describe and outline the research question, the goals of this thesis, and survey related work.

Chapter 3 considers the problem of using scalable multicast protocols to support on-demand download of large files from a single server to potentially large numbers of clients. Lower bounds were developed that indicate the best achievable performance. An optimized cyclic multicast protocol and two batching protocols, optimized for average and maximum client delay, were found to have significantly suboptimal performance over particular regions of the system design space, motivating the development of new hybrid protocols.

In the case of homogeneous clients, the best of the new practical protocols that focus on improving maximum client delay (*cyclic/cd,bot*) yielded results within 15% of optimal, in all scenarios considered. Similarly, the best of the new protocols designed to improve average client delay (*cyclic/rbd,cot*) yielded results within 20% of optimal. Both these protocols allow clients to begin listening to an on-going multicast if one is in progress at the times of their requests. Both protocols also achieve efficient batching of clients through use of a batching delay prior to the start of each multicast transmission and by limiting the transmission duration.

With the objective of minimizing the maximum client delay, *cyclic/cd,bot* uses a batching delay of fixed duration, and terminates each multicast transmission after delivering the full file or when a client completes reception of the file and there are no remaining listeners. In contrast, with the objective of minimizing the average client delay, *cyclic/rbd,cot* initiates each new multicast transmission only when the number of waiting clients reaches some minimum value. The multicast is terminated when the clients that were waiting at the beginning of the multicast have completed reception, and the number of newly arrived clients still listening to the multicast drops below some minimum value.

For heterogeneous clients, in all scenarios considered, the proposed *optimized sharing* protocol achieved within 25% of the optimal maximum client delay. This protocol uses multiple channels to deliver the file data, and an analytic model to estimate the optimal amount of data that each class of clients should retrieve from each channel. An interesting observation is that *optimized sharing* can substantially outperform *send as late as possible*, which is optimal in the homogenous environment.

Chapter 4 considers the replica selection problem for systems exploiting both replication and request aggregation. Two types of service aggregation are considered, batched service and fountain service. For each type of service aggregation, policy classes of differing complexities were compared within the context of a simple system model.

Results obtained by comparing optimal representatives under a simple cost model suggest that replica selection using dynamic system state information (rather than only proximities and average loads) can potentially yield large improvements in performance. Within the class of dynamic policies, use of deferred rather than at-arrival replica selection has the potential to yield further substantial performance improvements, although only for fairly narrow ranges of model parameter values. Finally, within the class of deferred, dynamic replica selection policies, "local state" policies appear able to achieve reasonably close to the best possible performance.

Chapter 5 considers systems in which clients are willing (or encouraged) to contribute with server capacity while downloading a file. A new approach is proposed in which adaptations of already existing peer-assisted download protocols, such as BitTorrent, are used to download pieces of a file in an order that allows streaming. Such protocols must include both (i) a piece selection strategy that effectively mediates the conflict between the goals of high piece diversity (achieved in BitTorrent using a rarest-first policy), and the in-order requirements of media file playback, and (ii) an on-line rule for deciding when playback can safely commence.

Using event based simulations it is shown that simple probabilistic piece selection policies, giving preference to earlier pieces, allow peers to begin playback well before the entire file is downloaded. These policies appear to effectively mediate the conflict between the goals of achieving high piece diversity (required for peers to effectively share pieces) and the in-order requirements of media file playback. While no on-line strategy for selecting startup delays is expected to give close to optimal startup delays (without significant chance of playback interruption), promising results are obtained using a simple rule, which requires the number of pieces to exceed some (small) threshold, and the rate at which in-order pieces are retrieved to allow playback to begin (if that rate was to be maintained).

6.2 Thesis Contributions

The following summarizes the main contributions of this work.

- *New single server bounds for the delivery of a single file*. Tight lower bounds on the average and maximum client delay for completely downloading a file, as a function of average server bandwidth used to serve requests for that file, for systems with homogenous clients. A lower bound algorithm on the average server bandwidth used to serve requests from heterogeneous clients with different client bandwidth and maximum delay constraints, given a sequence of such requests.
- *New near optimal single server download protocols for the delivery of a single file*. Relatively simple, near optimal (with regards to either average or maximum delay, given some average bandwidth usage), single server protocols for systems with homogenous clients. A protocol that achieves close to optimal for systems in which clients are heterogeneous, with classes of clients with different client bandwidth and delay constraints.
- *A classification and thorough performance comparison of policy classes in delivery (or service) systems using both service aggregation and replication techniques*. In particular, a simple cost model is developed in which (in many

cases optimal) representatives of classes of policies are compared, using both a batched and a fountain service model.

- *A new approach to achieve peer-assisted streaming is advocated*. Building upon already existing scalable download protocols, such as BitTorrent, the proposed protocol splits the file into multiple pieces, uses a probabilistic piece selection policy with bias towards pieces needed sooner, and a simple startup rule to determine when playback can safely commence.

6.3 Future Work

The following outlines some open research problems that may provide interesting future work.

- While the above work focuses on the delivery of a single file, an interesting problem is presented when considering the dynamics of systems in which more files are delivered concurrently. For example, a server with hundreds of files and a fixed server bandwidth (defining the maximum rate at which file data can be disseminated) may be required to, at each time instance, determine how much bandwidth (possibly none) to use for delivery of each file. An additional aspect that would be interesting to consider in such systems is the rate at which clients may leave the system (balk) before being fully served.
- Determining true optimal "on-line" performance for batched service policies, in systems utilizing both service aggregation and replication, remains an open problem. In fact, for the case of average delays, determining the optimal "off-line" performance (or some other good bound) on the achievable performance remains an open problem. Although the complexity of optimal "on-line" algorithms may be significant, it is believed that such algorithms may provide further insight into the desirable characteristics (and complexity) of real systems.
- To provide further insight into systems, utilizing both service aggregation and replication, more complex proximity and cost models could also be considered. For example, the distances to different replicas may be relatively different; causing the rate at which data is transferred (with TCP, for example) to depend on the client's distance from the replica at which it is being served.

- With heterogeneous client rates another interesting aspect is the affect of parallel downloading from multiple replica sites (each implementing service aggregation). In such systems, clients with greater client bandwidth may be served by more replicas than clients with lower bandwidth. With highly variable service times the impact of coordination among replicas may become important. For example, what is the cost of each replica acting independently, versus if all replicas are coordinated to achieve some common service goal?
- Developing a prototype implementation of BitTorrent-like streaming protocols on PlanetLab[1], could allow for wide area experiments giving further insight into the feasibility of using on-line rules to determine when to start playback, in heterogeneous wide area environments with clients operating under varying network conditions. A preliminary prototype is currently under development.
- To allow BitTorrent-like streaming protocols to better deal with time-varying reception rates, protocols could be implemented using layered media encoding techniques (e.g., [122, 142]). More advanced protocols could also take more system information into consideration. For example, to allow some minimum media quality an enterprise seeder may use information about the specific pieces possessed by individual peers, as well as their current play point, when determining which peers to serve, and which pieces to upload to each of these peers.

[1] PlanetLab, http://www.planet-lab.org/, August 2006.

References

[1] M. Abrams, C. R. Standridge, G. Abdulla, E. A. Fox, and S. Williams, "Removal Policies in Network Caches for World-Wide Web Documents", *Proc. ACM SIGCOMM '96*, Palo Alto, CA, Aug. 1996, pp. 293--305.

[2] S. Acharya, R. Alonso, M. J. Franklin, and S. B. Zdonik, "Broadcast Disks: Data Management for Asymmetric Communication Environments", *Proc. ACM SIGMOD '95*, San Jose, CA, May 1995, pp. 199--210.

[3] S. Acharya, M. J. Franklin, and S. B. Zdonik, "Balancing Push and Pull for Data Broadcast", *Proc. ACM SIGMOD '97*, Tucson, AZ, May 1997, pp. 183--194.

[4] S. Acharya, and S. Muthukrishnan, "Scheduling On-demand Broadcasts: New Metrics and Algorithms", *Proc. ACM/IEEE MOBICOM '98*, Dallas, TX, Oct. 1998, pp. 43--54.

[5] C. C. Aggarwal, J. L. Wolf, and P. S. Yu, "A Permutation-based Pyramid Broadcasting Scheme for Video-on-Demand Systems", *Proc. IEEE ICMCS '96*, Hiroshima, Japan, June 1996, pp. 118--126.

[6] C. C. Aggarwal, J. L. Wolf, and P. S. Yu, "On Optimal Batching Policies for Video-on-Demand Storage Servers", *Proc. IEEE ICMCS '96*, Hiroshima, Japan, June 1996, pp. 253--258.

[7] Akamai, "Fast Internet Content Delivery with FreeFlow", White paper, Apr. 2000.

[8] D. Aksoy, and M. Franklin, "RxW: A Scheduling Approach for Large-scale On-demand Data Broadcast", *IEEE/ACM Transactions on Networking 7*, 6 (Dec. 1999), pp. 846--860.

[9] J. M. Almeida, D. L. Eager, M. K. Vernon, and S. J. Wright, "Minimizing Delivery Cost in Scalable Streaming Content Distribution Systems", *IEEE Transactions on Multimedia 6*, 2 (Apr. 2004), pp. 356--365.

[10] K. C. Almeroth, M. H. Ammar, and Z. Fei, "Scalable Delivery of Web Pages Using Cyclic Best-effort (UDP) Multicast", *Proc. IEEE INFOCOM '98*, San Francisco, CA, Mar. 1998, pp. 1214--1221.

[11] K. C. Almeroth, "The Evolution of Multicast: From the Mbone to Interdomain Multicast to Internet2 Deployment", *IEEE Network (Special Issue on Multicasting) 14*, 1 (Jan/Feb. 2000), pp. 10--20.

[12] M. H. Ammar, and J. W. Wong. "On the Optimality of Cyclic Transmission in Teletext Systems", *IEEE Transactions on Communications 35*, 1 (Jan. 1987), pp. 68--73.

[13] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris, "Resilient Overlay Networks", *Proc. ACM SOSP '01*, Banff, AB, Canada, Oct. 2001, pp. 131--145.

[14] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris, "The Case for Resilient Overlay Networks", *Proc. HotOS-VIII '01*, Elmau/Oberbayern, Germany, May 2001, pp. 152--157.

[15] M. Andrews, B. Shepherd, A. Srinivasan, P. Winkler, and F. Zane, “Clustering and Server Selection Using Passive Monitoring”, *Proc. IEEE INFOCOM ’02*, New York, NY, June 2002, pp. 1717--1725.

[16] S. Annapureddy, C. Gkantsidis, and P. R. Rodriguez, “Providing Video-on-Demand using Peer-to-Peer Networks”, *Proc. Workshop on Internet Protocol TV* (*IPTV*) *’06*, Edinburgh, Scotland, May 2006.

[17] M. Balazinska, H. Balakrishnan, and D. Karger, “INS/Twine: A Scalable Peer-to-Peer Architecture for Intentional Resource Discovery”, *Proc. Pervasive ’02*, Zurich, Switzerland, Aug. 2002, pp. 195--210.

[18] T. Ballardie, “Core Based Trees (CBT version 2) Multicast Routing”, RFC 2189, IETF, Sept. 1997.

[19] T. Ballardie, P. Francis, and J. Crowcroft, “Core Based Trees (CBT) an Architecture for Scalable Multicast Routing”, *Proc. ACM SIGCOMM ’93*, San Francisco, CA, Sept. 1993, pp. 85--95.

[20] S. Banerjee, B. Bhattacharya, and C. Kommareddy, “Scalable Application Layer Multicast”, *Proc. ACM SIGCOMM ’02*, Pittsburgh, PA, Aug. 2002, pp. 205--217.

[21] A. Barbir, B. Cain, R. Nair, and O. Spatscheck, “Known Content Network (CN) Request-routing Mechanisms”, RFC 3568, IETF, July 2003.

[22] T. Bates, Y. Rekhter, R. Chandra, and D. Katz, “Multiprotocol Extensions for BGP-4”, RFC 2858, IETF, June 2000.

[23] D. Bertsekas, and R. Gallager, *Data Networks*, Prentice Hall, Englewood Cliffs, NJ, 1992.

[24] A. Bestavros, and S. Jin, “OSMOSIS: Scalable Delivery of Real-time Streaming Media in Ad-hoc Overlay Networks”, *Proc. ICDCS Workshops ’03*, Providence, RI, May 2003, pp. 214--219.

[25] A. R. Bharambe, C. Herley, V. N. Padmanabhan, “Analyzing and Improving a BitTorrent Network’s Performance Mechanisms”, *Proc. IEEE INFOCOM ’06*, Barcelona, Spain, Apr. 2006.

[26] S. Bhattacharyya, J. F. Kurose, D. Towsley, and R. Nagarajan, “Efficient Rate-controlled Bulk Data Transfer using Multiple Multicast Groups”, *Proc. IEEE INFOCOM ’98*, San Francisco, CA, Apr. 1998, pp. 1172--1179.

[27] Y. Birk, D. Crupnicoff, “A Multicast Transmission Schedule for Scalable Multi-rate Distribution of Bulk Data using Non-scalable Erasure-correcting Codes”, *Proc. IEEE INFOCOM ’03*, San Francisco, CA, Mar/Apr. 2003, pp. 1033--1043.

[28] J. W. Byers, J. Considine, M. Mitzenmacher, and S. Rost, “Informed Content Delivery Across Adaptive Overlay Networks”, *Proc. ACM SIGCOMM ’02*, Pittsburgh, PA, Aug. 2002, pp. 47--60.

[29] J. Byers, M. Frumin, G. Horn, M. Luby, M. Mitzenmacher, A. Roetter, and W. Shaver, “FLID-DL: Congestion Control for Layered Multicast”, *Proc. NGC ’00*, Palo Alto, CA, Nov. 2000, pp. 71--81.

[30] J. W. Byers, M. Luby, and M. Mitzenmacher, “Accessing Multiple Mirror Sites in Parallel: Using Tornado Codes to Speed Up Downloads”, *Proc. IEEE INFOCOM ’99*, New York, NY, Mar. 1999, pp. 275--283.

[31] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege, “A Digital Fountain Approach to Reliable Distribution of Bulk Data”, *Proc. ACM SIGCOMM ’98*, Vancouver, BC, Canada, Sept. 1998, pp 56--67.

[32] B. Cain, S. Deering, I. Kouvelas, B. Fenner, and A. Thyagarajan, “Internet Group Management Protocol, Version 3”, RFC 3376, IETF, Oct. 2002.

[33] C. W. Cameron, S. H. Low, and D. X. Wei, “High-density Model for Server Allocation and Placement”, *Proc. ACM SIGMETRICS ’02*, Marina Del Rey, CA, June 2002, pp. 152--159.

[34] P. Cao, and S. Irani, “Cost-aware WWW Proxy Caching Algorithms”, *Proc. USENIX ’97*, Monterey, CA, Dec. 1997, pp. 193--206.

[35] R. L. Carter, and M. E. Crovella, “Server Selection using Dynamic Path Characterization in Wide-area Networks”, *Proc. IEEE INFOCOM ’97*, Kobe, Japan, Apr. 1997, pp 1014--1021.

[36] S. W. Carter, and D. D. E. Long, “Improving Video-on-Demand Server Efficiency Through Stream Tapping”, *Proc. ICCCN ’97*, Las Vegas, CA, Sept. 1997, pp. 200--207.

[37] M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron, “Exploiting Network Proximity in Peer-to-Peer Overlay Networks”, Technical report MSR-TR-2002-82, Microsoft Research, May 2002.

[38] M. Castro, P. Druschel, A-M. Kermarrec and A. Rowstron, “SCRIBE: A Large-scale and Decentralized Application-level Multicast Infrastructure”, *IEEE Journal on Selected Areas in Communication 20*, 8 (Oct. 2002), pp. 1489--1499.

[39] M. Castro, P. Druschel, A. Rowstron, A-M. Kermarrec, A. Singh, and A. Nandi, “SplitStream: High-Bandwidth Multicast in Cooperative Environments”, *Proc. ACM SOSP ’03*, Bolton Landing, NY, Oct. 2003, pp. 298--313.

[40] M. Castro, M. B. Jones, A-M. Kermarrec, A. Rowstron, M. Theimer, H. Wang, and A. Wolman, “An Evaluation of Scalable Application-level Multicast Built Using Peer-to-Peer Overlays”, *Proc. IEEE INFOCOM ’03*, San Francisco, CA, Mar/Apr. 2003, pp. 1510--1520.

[41] R. C. Chalmers, and K. C. Almeroth, “On the Topology of Multicast Trees”, *IEEE/ACM Transactions on Networking 11*, 1 (Feb. 2003), pp. 153--165.

[42] Y. Chawathe, “Scattercast: An Adaptable Broadcast Distribution Framework”, *ACM Multimedia Systems Journal (Special Issue on Multimedia Distribution) 9*, 1 (July 2003), pp. 104--118.

[43] Y. Chawathe, S. McCanne, and E. Brewer, “RMX: Reliable Multicast in Heterogeneous Networks”, *Proc. IEEE INFOCOM ’00*, Tel Aviv, Israel, Mar. 2000, pp. 795--804.

[44] Y-H. Chu, S. G. Rao, S. Seshan, and H. Zhang, “Enabling Conferencing Applications on the Internet using an Overlay Multicast Architecture”, *Proc. ACM SIGCOMM ’01*, San Diego, CA, Aug. 2001, pp. 55--67.

[45] Y-H. Chu, S. G. Rao, and H. Zhang, “A Case for End System Multicast”, *Proc. ACM SIGMETRICS ’00*, Santa Clara, CA, June 2000, pp. 1--12.

[46] J. Chuang, and M. Sirbu, “Pricing Multicast Communication: A Cost Based Approach”, *Telecommunication Systems 17*, 3 (July 2001), pp. 281--297.

[47] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, “Freenet: A Distributed Anonymous Information Storage and Retrieval System”, *Proc. International Workshop on Design Issues in Anonymity and Unobservability ’00*, Berkeley, CA, July 2000, pp. 46--66.

[48] B Cohen, “Incentives Build Robustness in BitTorrent”, *Proc. Workshop on Economics of Peer-to-Peer Systems ’03*, Berkeley, CA, June 2003.

[49] E. Cohen, A. Fiat, and H. Kaplan, “A Case for Associative Peer to Peer Overlays”, *Proc. HotNets-I ’02*, Princeton, NJ, Oct. 2002, pp. 95--100.

[50] E. Cohen, A. Fiat, and H. Kaplan, “Associative Search in Peer to Peer Networks: Harnessing Latent Semantics”, *Proc. INFOCOM ’03*, San Francisco, CA, Mar/Apr. 2003, pp. 1261--1271.

[51] E. Cohen, and S. Shenker, “Replication Strategies in Unstructured Peer-to-Peer Networks”, *Proc. ACM SIGCOMM ’02*, Pittsburgh, PA, Aug. 2002, pp. 177--190.

[52] L. H. M. K. Costa, S. Fdida, and O. C. M. B. Duarte, “Hop-by-Hop Multicast Routing Protocol”, *Proc. ACM SIGCOMM ’01*, San Diego, CA, Aug. 2001, pp. 249--259.

[53] Y. Cui, B. Li, and K. Nahrstedt, “oStream: Asynchronous Streaming Multicast in Application-layer Overlay Networks”, *IEEE Journal on Selected Areas in Communications (Special Issue on Recent Advances in Service Overlays) 22*, 1 (Jan. 2004), pp. 91--106.

[54] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, “Vivaldi: a Decentralized Network Coordinate System”, *Proc. ACM SIGCOMM ’04*, Portland, OR, Aug/Sept. 2004, pp. 15--26.

[55] F. Dabek, M.F. Kaashoek, D. Karger, R. Morris, and I. Stoica, “Wide-area Cooperative Storage with CFS”, *Proc. ACM SOSP ’01*, Banff, AB, Canada, Oct. 2001, pp. 202--215.

[56] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica, “Towards a Common API for Structured Peer-to-Peer Overlays”, *Proc. IPTPS ’03*, Berkeley, CA, Feb. 2003, pp. 33--44.

[57] A. Dan , D. Sitaram, and P. Shahabuddin, “Scheduling Policies for an On-demand Video Server with Batching”, *Proc. ACM Multimedia ’94*, San Francisco, CA, Oct. 1994, pp. 15--23.

[58] M. Day, B. Cain, G. Tomlinson, and P. Rzewski, “A Model for Content Internetworking”, RFC 3466, IETF, Feb. 2003.

[59] S. Deering, and D. R. Cheriton, “Multicast Routing in Datagram Internetworks and Extended LANs”, *ACM Transactions on Computer Systems 8*, 2 (May 1990), pp. 85--111.

[60] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C-G. Liu, and L. Wei, “An Architecture for Wide Area Multicast Routing”, *Proc. ACM SIGCOMM ’94*, London, UK, Sept. 1994, pp. 126--135.

[61] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, and L. Wei, “The PIM Architecture for Wide-area Multicast Routing”, *IEEE/ACM Transactions on Networking 4*, 2 (Apr. 1996), pp.153--162.

[62] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl “Globally Distributed Content Delivery”, *IEEE Internet Computing 6*, 5 (Sept/Oct. 2002), pp. 50--58.

[63] C. Diot, L. Guiliano, R. Rockell, J. Meylor, D. Meyer, G. Shepherd, and B. Haberman, “An Overview of Source-Specific Multicast (SSM)”, RFC 3569, IETF, July 2003.

[64] C. Diot, B. N. Levine, B. Liles, H. Kassem, and D. Balensiefen, “Deployment Issues for the IP Multicast Service and Architecture”, *IEEE Network Magazine (Special Issue on Multicasting) 14*, 1, (Jan/Feb. 2000), pp. 78--88.

[65] C. Dovrolis, P. Ramanathan, and D. Moore, “What do Packet Dispersion Techniques Measure?”, *Proc. IEEE INFOCOM ’01*, Anchorage, AK, Apr. 2001, pp. 905--914.

[66] H. D. Dykeman, M. H. Ammar, and J. W. Wong, “Scheduling Algorithms for Videotex Systems under Broadcast Delivery”, *Proc. ICC ’86*, Toronto, ON, Canada, June 1986, pp. 1847--1851.

[67] D. L. Eager, M. K. Vernon, and J. Zahorjan, “Optimal and Efficient Merging Schedules for Video-on-Demand Servers”, *Proc. ACM Multimedia ’99*, Orlando, FL, Nov. 1999, pp. 199--202.

[68] D. L. Eager, M. K. Vernon, and J. Zahorjan, “Bandwidth Skimming: A Technique for Cost-effective Video-on-Demand”, *Proc. MMCN ’00*, San Jose, CA, Jan. 2000, pp. 206--215.

[69] D. L. Eager, M. K. Vernon, and J. Zahorjan, “Minimizing Bandwidth Requirements for On-demand Data Delivery”, *IEEE Transactions on Knowledge and Data Engineering 13*, 5 (Sept/Oct. 2001), pp. 742--757.

[70] H. Eriksson, “MBONE: The Multicast Backbone”, *Communications of the ACM 37*, 8 (Aug. 1994), pp. 54--60.

[71] L. Fan, P. Cao, J. Almeida, and A. Broder, “Summary Cache: A Scalable Wide-area Web Cache Sharing Protocol”, *Proc. ACM SIGCOMM ’98*, Vancouver, BC, Canada, Sept. 1998, pp. 254--265.

[72] Z. Fei, M. H. Ammar, and E. W. Zegura, “Multicast Server Selection: Problems, Complexity and Solutions”, *IEEE Journal on Selected Areas in Communications 20*, 7 (Sept. 2002), pp. 1399--1413.

[73] S. Floyd, “Connections with Multiple Congested Gateways in Packet-switched Networks Part 1: One-way Traffic”, *Computer Communication Review 21*, 5 (Oct. 1991), pp. 30--47.

[74] P. Francis, “Yoid: Your Own Internet Distribution”, Technical Report, ACIRI, Berkeley, CA, March 2001.

[75] S. Ganguly, A. Saxena, S. Bhatnagar, R. Izmailov, and S. Banerjee, “Fast Replication in Content Distribution Overlays”, *Proc. IEEE INFOCOM ’05*, Miami, FL, Mar. 2005, pp. 2246--2256.

[76] L. Gao, and D. Towsley, “Supplying Instantaneous Video-on-Demand Services using Controlled Multicast”, *Proc. ICMCS ’99*, Florence, Italy, June 1999, pp. 117--121.

[77] C. Gkantsidis, M. Ammar, and E. Zegura, “On the Effect of Large-scale Deployment of Parallel Downloading”, *Proc. WIAPP ’03*, San Jose, CA, June 2003, pp. 79--89.

[78] C. Gkantsidis, and P. R. Rodriguez, “Network Coding for Large Scale Content Distribution”, *Proc. IEEE INFOCOM ’05*, Miami, FL, Mar. 2005, pp. 2235--2245.

[79] L. Golubchik, J. C-S. Lui, and R. R. Muntz, “Adaptive Piggybacking: a Novel Technique for Data Sharing in Video-on-Demand Storage Servers”, *ACM Multimedia Systems Journal 4*, 3 (June 1996), pp. 140--155.

[80] M. Green, B. Cain, G. Tomlinson, M. Speer, P. Rzewski, and S. Thomas, “Content Internetworking Architectural Overview”, Internet Draft, IETF, June 2002.

[81] M. Guo, M. H. Ammar, and E. W. Zegura, “Selecting among Replicated Batching Video-on-Demand Servers”, *Proc. NOSSDAV ’02*, Miami Beach, FL, May 2002, pp. 155--163.

[82] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang, “Measurement, Analysis, and Modeling of BitTorrent-like Systems”, *Proc. IMC ’05*, Berkeley, CA, Oct. 2005, pp. 35--48.

[83] S. Hameed, and N. H. Vaidya, “Log-Time Algorithms for Scheduling Single and Multiple Channel Data Broadcast”, *Proc. ACM/IEEE MOBICOM ’97*, Budapest, Hungary, Sept. 1997, pp. 90--99.

[84] S. Hameed, and N. H. Vaidya, “Efficient Algorithms for Scheduling Data Broadcast”, *Wireless Networks 5*, 3 (May 1999), pp. 183--193.

[85] M. Hefeeda, A. Habib, B. Botev, D. Xu, B. Bhargava, “PROMISE: Peer-to-Peer Media Streaming using CollectCast”, *Proc. ACM Multimedia ’03*, Berkeley, CA, Nov. 2003, pp. 45--54.

[86] X. Hei, C. Liang, J. Liang, Y. Liu, and K. W. Ross, “A Measurement Study of a Large-scale P2P IPTV System”, Technical Report, Department of Computer and Information Science, Polytechnic University, Brooklyn, NY, Oct. 2006.

[87] H. W. Holbrook, and D. R. Cheriton, “IP Multicast Channels: EXPRESS Support for Large-scale Single-source Applications”, *Proc. ACM SIGCOMM ’99*, Cambridge, MA, Sept. 1999, pp. 65--78.

[88] A. Hu. “Video-on-Demand Broadcasting Protocols: A Comprehensive Study”, *Proc. IEEE INFOCOM ’01*, Anchorage, AK, April, 2001, pp. 508--517.

[89] K. A. Hua, Y. Cai, and S. Sheu, “Patching: A Multicast Technique for True Video-on-Demand Services”, *Proc. ACM Multimedia ’98*, Bristol, UK, Sept. 1998, pp. 191--200.

[90] K. A. Hua, and S. Sheu, “Skyscraper Broadcasting: A New Broadcasting Scheme for Metropolitan Video-on-Demand Systems”, *Proc. ACM SIGCOMM ’97*, Cannes, France, Sept. 1997, pp. 89--100.

[91] M. Izal, G. Urvoy-Keller, E. W. Biersack, P. Felber, A. A. Hamra, and L. Garces-Erice, “Dissecting Bittorrent: Five Months in a Torrent’s Lifetime”, *Proc. PAM ’04*, Antibes Juan-les-Pins, France, Apr. 2004, pp. 1--11.

[92] M. Jain, and C. Dovrolis, “End-to-End Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput”, *Proc. ACM SIGCOMM ’02*, Pittsburgh, PA, Aug. 2002, pp. 295--308.

[93] S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang, “On the Placement of Internet Instrumentation”, *Proc. IEEE INFOCOM ’00*, Tel Aviv, Israel, Mar. 2000, pp. 295--304.

[94] S. Jamin, C. Jiu, A. Kurc, D. Raz, and Y. Shavitt, “Constrained Mirror Placement on the Internet”, *Proc. IEEE INFOCOM ’01*, Anchorage, AK, Apr. 2001, pp. 31--40.

[95] J. Jannotti, D. Gifford, K. Johnson, M. Kaashoek, and J. O'Toole, “Overcast: Reliable Multicasting with an Overlay Network”, *Proc. OSDI ’00*, San Diego, CA, Oct. 2000, pp. 197--212.

[96] K. L. Johnson, J. F. Carr, M. S. Day, and F. Kaashoek, “The Measured Performance of Content Distribution Networks”, *Computer Communications 24*, 2 (Feb. 2001), pp. 202--206.

[97] J. Kangasharju, K. W. Ross, J. W. Roberts, “Performance Evaluation of Redirection Schemes in Content Distribution Networks”, *Computer Communications, 24*, 2 (Feb. 2001), pp. 207--214.

[98] T. Karagiannis, P. Rodriguez, and K. Papagiannaki, “Should Internet Service Providers Fear Peer-assisted Content Distribution?”, *Proc. IMC ’05*, Berkeley, CA, Oct. 2005, pp. 63--76.

[99] P. Karbhari, E. Zegura, and M. Ammar, “Multipoint-to-Point Session Fairness in the Internet”, *Proc. IEEE INFOCOM ’03*, San Francisco, CA, Mar/Apr. 2003, pp. 207--217.

[100] C. Kenyon, N. Schabanel, and N. Young, “Polynomial-time Approximation Scheme for Data Broadcast”, *Proc. STOC ’00*, Portland, OR, May 2000, pp. 659--666.

[101] S. G. M. Koo, C. Rosenberg, and D. Xu, "Analysis of Parallel Downloading for Large File Distribution, *Proc. FTDCS '03*, San Juan, Puerto Rico, May 2003, pp. 128--135.

[102] D. Kozic, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: High Bandwidth Data dissemination Using and Overlay Mesh", *Proc. ACM SOSP '03*, Bolton Landing, NY, Oct. 2003, pp. 282--297.

[103] B. Krishnamurthy, C. Wills, and Y. Zhang, "On the use and Performance of Content Distribution Networks", *Proc. IMW '01*, San Francisco, CA, Nov. 2001, pp. 169--182.

[104] P. Krishnan, D. Raz, and Y. Shavitt, "The Cache Location Problem", *IEEE/ACM Transactions on Networking 8*, 5 (Oct. 2000), pp. 568--582.

[105] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "Oceanstore: An Architecture for Global-scale Persistent Storage", *Proc. ASPLOS '00*, Cambridge, MA, Nov. 2000, pp. 190--201.

[106] S. Kumar, P. Radoslavov, D. Thaler, C. Alaettinoglu, D. Estrin, and M. Handley, "The MASC/BGMP Architecture for Inter-domain Multicast Routing", *Proc. ACM SIGCOMM '98*, Vancouver, BC, Canada, Sept. 1998, pp. 93--104.

[107] G. Kwon, and J. Byers, "Smooth Multirate Multicast Congestion Control", *Proc. IEEE INFOCOM '03*, San Francisco, CA, Mar/Apr. 2003, pp. 1022--1032.

[108] G-I. Kwon, and J. W. Byers, "ROMA: Reliable Overlay Multicast with Loosely Coupled TCP Connections", *Proc. IEEE INFOCOM '04*, Hong Kong, China, Mar. 2004, pp. 385--395.

[109] T. V. Lakshman, and U. Madhow, "The Performance of TCP/IP for Networks with High Bandwidth-delay Products and Random Loss", *IEEE/ACM Transactions on Networking 5*, 3 (June 1997), pp. 336--350.

[110] A. Legout, G. Urvoy-Keller, and P. Michiardi, "Understanding BitTorrent: An Experimental Perspective", Technical Report (inria-00000156, version 3 - 9 November 2005), INRIA, Sophia Antipolis, France, Nov. 2005.

[111] A. Legout, G. Urvoy-Keller, and P. Michiardi, "Rarest First and Choke Algorithms Are Enough", *Proc. IMC '06*, Oct. 2006.

[112] X. Liao, H. Jin, Y. Liu, L. M. Ni, and D. Deng, "AnySee: Peer-to-Peer Live Streaming", *Proc. IEEE INFOCOM '06*, Barcelona, Spain, Apr. 2006.

[113] M. Luby, V. K. Goyal, S. Skaria, and G. B. Horn, "Wave and Equation Based Rate Control Using Multicast Round Trip Time", *Proc. ACM SIGCOMM '02*, Pittsburgh, PA, Aug. 2002, pp. 191--204.

[114] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, D. Spielman, and V. Stemann, "Practical Loss-resilient Codes", *Proc. ACM STOC '97*, El Paso, TX, May 1997, pp. 150--159.

[115] M. Luby, L. Vicisano, J. Gemmell, L. Rizzo, M. Handley, and J. Crowcroft, "The use of Forward Error Correction (FEC) in Reliable Multicast", RFC 3453, IETF, Dec. 2002.

[116] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, “Search and Replication in Unstructured Peer-to-Peer Networks”, *Proc. ICS ’02*, New York, NY, June 2002, pp. 84--95.

[117] Q. Lv, S. Ratnasamy, and S. Shenker, “Can Heterogeneity Make Gnutella Scalable?”, *Proc. IPTPS ’02*, Cambridge, MA, Mar. 2002, pp. 94--103.

[118] N. Magharei, and R. Rejaie, “Adaptive Receiver-driven Streaming from Multiple Senders”, *Multimedia Systems 11*, 6 (June 2006), pp. 550--567.

[119] A. Mahanti, D. L. Eager, M. K. Vernon, and D. Sundaram-Stukel, “Scalable On-demand Media Streaming with Packet Loss Recovery”, *Proc ACM SIGCOMM ’01*, San Diego, CA, Aug. 2001, pp. 97--108.

[120] L. Massoulie, and M. Vojnovic, “Coupon Replication Systems”, *Proc. ACM SIGMETRICS ’05*, Banff, Canada, June 2005, pp. 2--13.

[121] P. Maymounkov, and D. Mazières, “Rateless Codes and Big Downloads”, *Proc. IPTPS ’03*, Berkeley, CA, Feb. 2003, pp. 247--255.

[122] S. McCanne, V. Jacobson, and M. Vetterli, “Receiver-driven Layered Multicast”, *Proc. ACM SIGCOMM ’96*, Stanford, CA, Aug. 1996, pp. 117--130.

[123] J. Moy, “MOSPF: Analysis and Experience”, RFC 1585, IETF, Mar. 1994.

[124] A. Oram (editor), *Peer-to-Peer Harnessing the Power of Disruptive Technologies*, O’Reilly & Associates, Sebastopol, CA, Mar. 2001.

[125] V. N. Padmanabhan, H. J. Wang, and P. A. Chou, “Resilient Peer-to-Peer Streaming”, *Proc. ICNP ’03*, Atlanta, GA, Nov. 2003, pp. 16--27.

[126] K. Park, and V. S. Pai, “Deploying Large File Transfer on an HTTP Content Distribution Network”, *Proc. WORLDS ’04*, San Francisco, CA, Dec. 2004.

[127] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel, “ALMI: An Application Level Multicast Infrastructure”, *Proc. USITS ’01*, San Francisco, CA, Mar. 2001, pp. 49--60.

[128] G. Phillips, S. Shenker, and H. Tangmunarunkit, “Scaling of Multicast Trees: Comments on the Chuang-Sirbu Scaling Law”, *Proc. ACM SIGCOMM ’99*, Cambridge, MA, Aug. 1999, pp. 41--51.

[129] S. Philopoulos, and M. Maheswaran, “Experimental Study of Parallel Downloading Schemes for Internet Mirror Sites”, *Proc. PDCS ’01*, Richardson, TX, Aug. 2001, pp. 44--48.

[130] T. Piotrowski, S. Banerjee, S. Bhatnagar, S. Ganguly, and R. Izmailov, “Peer-to-Peer Streaming of Stored Media: the Indirect Approach”, *Proc. SIGMETRICS ’06*, Saint-Malo, France, June 2006, pp. 371--372.

[131] C. G. Plaxton, R. Rajaraman, and A. W. Richa, “Accessing Nearby Copies of Replicated Objects in a Distributed Environment”, *Theory of Computing Systems 32*, 3 (June 1999), pp. 241--280.

[132] J. A. Pouwelse, P. Garbacki, D. H. J. Epema, H. J. Sips, “The Bittorrent P2P File-sharing System: Measurements and Analysis”, *Proc. IPTPS ’05*, Feb. 2005, pp. 205--216.

[133] L. Qiu, V. N. Padmanabhan, and G. Voelker, “On the Placement of Web Server Replicas”, *Proc. IEEE INFOCOM ’01*, Anchorage, AK, Apr. 2001, pp. 1587--1596.

[134] D. Qiu, and R. Srikant, “Modeling and Performance Analysis of BitTorrent-like Peer-to-Peer Networks”, *Proc. SIGCOMM ’04*, Portland, OR, Aug/Sept. 2004, pp. 367--378.

[135] P. Radoslavov, R. Govindan, D. Estrin, “Topology-informed Internet Replica Placement”, *Computer Communications 25*, 4 (Mar. 2002), pp. 384--392.

[136] B. Radunovic, and J. Y. Le Boudec, “A Unified Framework for Max-min and Min-max Fairness with Applications”, *Proc. Allerton Conf. on Communication, Control, and Computing ’02*, Allerton, IL, Oct. 2002.

[137] P. Rajvaidya, and K. Almeroth, “Analysis of Routing Characteristics in the Multicast Infrastructure”, *Proc. IEEE INFOCOM ’03*, San Francisco, CA, Mar/Apr. 2003, pp. 1532--1542.

[138] S. Ratnasamy, A. Ermolinskiy, S. Shenker, “Revisiting IP Multicast”, *Proc. ACM SIGCOMM ’06*, Pisa, Italy, Sept. 2006, pp. 15--26.

[139] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, “Application-level Multicast using Content-addressable Networks”, *Proc. NGC ’01*, London, UK, Nov. 2001, pp. 14--29.

[140] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, “Topologically-aware Overlay Construction and Server Selection”, *Proc. IEEE INFOCOM ’02,* New York, NY, June 2002, pp. 1190--1199.

[141] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, “A Scalable Content Addressable Network”, *Proc. ACM SIGCOMM ’01*, San Diego, CA, Aug. 2001, pp. 161--172.

[142] R. Rejaie, D. Estrin, and M. Handley, “Quality Adaptation for Congestion Controlled Video Playback over the Internet”, *Proc. ACM SIGCOMM ’99*, Cambridge, MA, Sept. 1999, pp. 189--200.

[143] R. Rejaie, and A. Ortega, “PALS: Peer-to-Peer Adaptive Layered Streaming”, *Proc. NOSSDAV ’03*, Monterey, CA, June 2003, pp. 153--161.

[144] Y. Rekhter, and T. Li, “A Border Gateway Protocol 4 (BGP-4)”, RFC 1771, IETF, March 1995.

[145] M. Ripeanu, and I. Foster, “Mapping the Gnutella Network: Macroscopic Properties of Large-scale Peer-to-Peer Systems”, *Proc. IPTPS ’02*, Cambridge, MA, Mar. 2002, pp. 85--93.

[146] L. Rizzo and L. Vicisano, “A Reliable Multicast Data Distribution Protocol Based on Software FEC Techniques”, *Proc. HPCS ’97*, Chalkidiki, Greece, June 1997, pp. 116--125.

[147] L. Rizzo, and L. Vicisano, “Replacement Policies for a Proxy Cache”, *IEEE/ACM Transactions on Networking 8*, 2 (Apr. 2000), pp. 158--170.

[148] L. Rizzo, "Effective Erasure Codes for Reliable Computer Communication Protocols", *ACM Computer Communication Review 27,* 2 (Apr. 1997), pp. 24--36.

[149] P. Rodriguez, and E. W. Biersack, "Dynamic Parallel-access to Replicated Content in the Internet", *IEEE/ACM Transactions on Networking 10,* 4 (Aug. 2002), pp. 455--465.

[150] S. Rost, J. Byers, and A. Bestavros, "The Cyclone Server Architecture: Streamlining Delivery of Popular Content", *Proc. WCW '01,* Boston, MA, June 2001, pp. 147--163.

[151] A. Rowstron, and P. Druschel, "Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-Peer Systems", *Proc. Middleware '01*, Heidelberg, Germany, Nov. 2001, pp. 329--350.

[152] A. Rowstron, and P. Druschel, "Storage Management and Caching in PAST, a Large-scale, Persistent Peer-to-Peer Storage Utility", *Proc. ACM SOSP '01*, Banff, AB, Canada, Oct. 2001, pp. 188--201.

[153] A. I. T. Rowstron, A-M. Kermarrec, M. Castro, and P. Druschel, "SCRIBE: The Design of a Large-scale Event Notification Infrastructure", *Proc. NGC '01*, London, UK, Nov. 2001, pp. 30--43.

[154] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-End Arguments in System Design", *ACM Transactions in Computer Systems 2*, 4 (Nov. 1984), pp. 277--288.

[155] K. Sarac and K. Almeroth, "Monitoring IP Multicast in the Internet: Recent Advances and Ongoing Challenges", *IEEE Communications Magazine 43*, 10 (Oct. 2005), pp. 85--91.

[156] S. Saroiu, K. P. Gummadi, and S. D. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems", *Proc. MMCN '02*, San Jose, CA, Jan. 2002, pp. 156--170.

[157] S. Saroiu, K. P. Gummadi, R. J. Dunn, S. D. Gribble, and H. M. Levy, "An Analysis of Internet Content Delivery Systems", *Proc. OSDI '02,* Boston, MA, Dec. 2002, pp. 315--327.

[158] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, G. Voelker, and J. Zahorjan, "Detour: a Case for Informed Internet Routing and Transport", *IEEE Micro 19*, 1 (Jan. 1999), pp. 50--59.

[159] S. Savage, A. Collins, E. Hoffman, J. Snell, and T. Anderson, "The End-to-End Effects of Internet Path Selection", *Proc. ACM SIGCOMM '99*, Cambridge, MA, Aug. 1999, pp. 289--299.

[160] P. Savola, "Overview of the Internet multicast Routing Architecture", Internet Draft (draft-ietf-mboned-routingarch-06.txt), IETF, Aug. 2006.

[161] A. Sharma, A. Bestavros, and I. Matta, "dPAM: A Distributed Prefetching Protocol for Scalable Asynchronous Multicast in P2P Systems", *Proc IEEE INFOCOM '05*, Miami, FL, Mar. 2005, pp. 1139--1150.

[162] R. Sherwood, R. Braud, and B. Bhattacharjee, “Slurpie: A Cooperative Bulk Data Transfer Protocol”, *Proc. IEEE INFOCOM ’04*, Hong Kong, China, Mar. 2004, pp. 941--951.

[163] S. Sheu, K. A. Hua, W. Tavanapong, “Chaining: A Generalized Batching Technique for Video-on-Demand”, *Proc ICMCS ’97*, Ottawa, ON, Canada, pp. 110--117.

[164] A. Shokrollahi, “Raptor Codes”, Technical Report DF2003-06-001, Digital Fountain Inc., June 2003.

[165] J. Song, C. Sha, and H. Zhu, “Nash Equilibria in Parallel Downloading with Multiple Clients”, *Proc. ICDCS ’04*, Tokyo, Japan, Mar. 2004, pp. 94--101.

[166] Stardust, “Content Networking and Edge Service: Leveraging the Internet for Profit”, White paper, Stardust.com Inc., Los Gatos, CA, Sept. 2001.

[167] K. Stathatos, N. Roussopoulos, and J. Baras, “Adaptive Data Broadcast in Hybrid Networks”, *Proc. VLDB ’97*, Athens, Greece, Sept. 1997, pp. 326--335.

[168] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana, “Internet Indirection Infrastructure”, *Proc. ACM SIGCOMM ’02*, Pittsburgh, PA, Aug. 2002, pp. 73--88.

[169] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, “Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications”, *Proc. ACM SIGCOMM ’01*, San Diego, CA, Aug. 2001, pp. 149--160.

[170] I. Stoica, T. S. E. Ng, and H. Zhang, “REUNITE: A recursive unicast approach to multicast”, *Proc. IEEE INFOCOM ’00*, Tel Aviv, Israel, Mar. 2000, pp. 1644--1653.

[171] H. Tan, D. L. Eager, and M. K. Vernon, “Delimiting the Range of Effectiveness of Scalable On-demand Streaming”, *Proc. Performance ’02,* Rome, Italy, Sept. 2002, pp. 387--410.

[172] C. Tang, Z. Xu, and M. Mahalingam, “pSearch: Information Retrieval in Structured Overlays”, *Proc. HotNets-I ’02*, Princeton, NJ, Oct. 2002, pp. 89--94.

[173] A. K. Tsiolis, and M. K. Vernon, “Group Guaranteed Channel Capacity in Multimedia Storage Servers”, *Proc. ACM SIGMETRICS ’97*, Seattle, WA, June 1997, pp. 285--297.

[174] N. H. Vaidya, and S. Hameed, ”Scheduling Data Broadcast in Asymmetric Communication Environments”, *Wireless Networks 5*, 3 (May 1999), pp.171--182.

[175] P. Van Mieghem, G. Hooghiemstra, and R. van der Hoftstrad, “On the Efficiency of Multicast”, *IEEE/ACM Transactions on Networking 9*, 6 (Dec. 2001), pp. 719--732.

[176] L. Vicisano, L. Rizzo, and J. Crowcroft, “TCP-like Congestion Control for Layered Video Multicast Data Transfer”, *Proc. IEEE INFOCOM ’98*, San Francisco, CA, Apr. 1998, pp. 996--1003.

[177] S. Viswanathan, and T. Imielinski, "Metropolitan Area Video-on-Demand Service using Pyramid Broadcasting", *Multimedia Systems 4*, 4 (Aug. 1996), pp. 197--208.

[178] D. Waitzman, C. Partridge, and S. Deering, "Distance Vector Multicast Routing Protocol", RFC 1075, IETF, Nov. 1988.

[179] L. Wang, V. Pai, and L. Peterson, "The Effectiveness of Request Redirection on CDN Robustness", *Proc. OSDI '02*, Boston, MA, Dec. 2002, pp. 345--360.

[180] J. L. Wolf, M. S. Squillante, J. Turek, P. S. Yu, and J. Sethuraman, "Scheduling Algorithms for the Broadcast Delivery of Digital Products", *IEEE Transactions on Knowledge and Data Engineering 13*, 5 (Sept/Oct. 2001), pp. 721--741.

[181] A. Wolman, G. M. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. M. Levy, "On the Scale and Performance of Cooperative Web Proxy Caching", *Proc. ACM SOSP '99*, Kiawah Island Resort, SC, Dec. 1999, pp. 16--31.

[182] J. W. Wong, "Broadcast Delivery", *Proc. of the IEEE 76*, 12 (Dec. 1988), pp. 1566--1577.

[183] X. R. Wu, and A. A. Chien, "A Distributed Algorithm for Max-min Fair Bandwidth Sharing", Technical Report, Department of Computer Science, University of California (UCSD), San Diego, CA, Oct. 2005.

[184] B. Yang, and H. Garcia-Molina, "Improving Search in Peer-to-Peer Networks", *Proc. ICDCS '02*, Vienna, Austria, July 2002, pp. 5--14.

[185] X. Yang, and G. de Veciana, "Service Capacity of Peer to Peer Networks", *Proc. IEEE INFOCOM '04*, Hong Kong, China, Mar. 2004, pp. 2242--2252.

[186] E. W. Zegura, M. H. Ammar, Z. Fei, and S. Bhattacharjee, "Application-layer Anycasting: a Server Selection Architecture and use in a Replicated Web Service", *IEEE/ACM Transactions on Networking 8*, 4 (Aug. 2000), pp. 455--466.

[187] A. Zeitoun, H. Jamjoom, and M. El-Gendy, "Scalable Parallel-access for Mirrored Servers", *Proc. IASTED Applied Informatics '02*, Innsbruck, Austria, Feb. 2002.

[188] H. Zhang, A. Goel, and R. Govindan, "Using the Small-world Model to Improve Freenet Performance", *Proc. IEEE INFOCOM '02*, New York, NY, June 2002, pp. 1228--1237.

[189] B. Zhang, S. Jamin, and L. Zhang, "Host Multicast: A Framework for Delivering Multicast to End Users", *Proc. IEEE INFOCOM '02*, New York, NY, June 2002, pp. 1366--1375.

[190] X. Zhang, J. Liu, B. Li, T-S. P. Yum, "CoolStreaming/DONet: A Data-driven Overlay Network for Peer-to-Peer Live Media Streaming", *Proc. IEEE INFOCOM '05*, Miami, FL, Mar. 2005, pp. 2102--2111.

[191] M. Zhang, L. Zhao, Y. Tang, J-G. Luo, and S-Q. Yang, "Large-scale Live Media Streaming over Peer-to-Peer Networks through Global Internet", *Proc. Workshop on Advances in Peer-to-Peer Multimedia Streaming '05*, Hilton, Singapore, Nov. 2005, pp. 21--28.

[192] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, “Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing”, Technical Report UCB/CSD-01-1141, University of California, Berkeley, CA, Apr. 2000.

[193] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. Katz, and J. Kubiatowicz, “Bayeux: An Architecture for Scalable and Fault-tolerant Wide-area Data Dissemination”, *Proc. NOSSDAV ’01*, Port Jefferson, NY, June 2001, pp. 11--20.

Appendix A

Proof of Single Server Heterogeneous Lower Bound

This appendix provides a proof of the *heterogonous lower bound* computed using the algorithm in Figure 3.10; i.e., that $B_j^{hlb} \le B_j$ for any realizable protocol. In fact, this proof actually prove a stronger result, by considering a more general algorithm in which the expression giving x_j^{hlb} in Figure 3.10 is replaced by $L - y_{j-1,j}^{hlb} - \varepsilon_j$, where the ε_j, $1 \le j \le K$, can be chosen to be any values such that $L - y_{j-1,j}^{hlb} \ge \varepsilon_j \ge 0$.

The proof that this more general algorithm yields a lower bound on B_j uses strong induction on *j*. As each client receives an amount of data equal to the file size *L*, in the case of just a single request $B_1^{hlb} = L - \varepsilon_1 \le L = B_1$, thus establishing the induction basis. Now, assume that $B_j^{hlb} \le B_j$, $B_{j-1}^{hlb} \le B_{j-1}$, …, $B_1^{hlb} \le B_1$, for some $j \ge 1$. The proof shows that $B_{j+1}^{hlb} \le B_{j+1}$ by establishing that

$$B_k^{hlb} + L - y_{k,j+1}^{hlb} \le B_k + L - y_{k,j+1}, \tag{A.1}$$

for k = 1, 2, …, *j*. Note that for $k = j$, relation (A.1) implies that

$$\begin{aligned} B_{j+1}^{hlb} &= B_j^{hlb} + x_{j+1}^{hlb} = B_j^{hlb} + \left(L - y_{j,j+1}^{hlb} - \varepsilon_{j+1}\right) \\ &\le B_j^{hlb} + L - y_{j,j+1}^{hlb} \le B_j + L - y_{j,j+1} = B_j + x_{j+1} = B_{j+1}. \end{aligned} \tag{A.2}$$

Relation (A.1) is proven by strong induction on *k*. For k = 1, since $B_1^{hlb} = L - \varepsilon_1$, $B_1 = L_1$, $y_{1,j+1}^{hlb} = x_{1,j+1}^{hlb}$, and $y_{1,j+1} = x_{1,j+1}$, relation (A.1) is equivalent to $x_{1,j+1} \le x_{1,j+1}^{hlb} + \varepsilon_1$. If $T_{j+1}^A \ge T_1^D$, $x_{1,j+1} = x_{1,j+1}^{hlb} = 0$ and the relation holds. Otherwise, using the expression giving $x_{j,i}^{hlb}$ in Figure 3.10, $x_{1,j+1} \le x_{1,j+1}^{hlb} + \varepsilon_1$ is equivalent to

$$x_{1,j+1} \le \min\{L - \varepsilon_1,\ \ L,\ \ b_{c(j+1)}T_{1,j+1},\ \ b_{c(j+1)}T_{1,1}\} + \varepsilon_1 . \tag{A.3}$$

Since the amount of data received by the request 1 client from transmissions also received by the request *j*+1 client can be at most *L*, and at most $b_{c(j+1)}$ times the period over which such transmissions can occur, this establishes the induction basis.

Suppose now that relation (A.1) holds for *k*, *k*-1, …, 1, for some *k* such that $j > k \ge 1$, and consider the relation for *k*+1. If $T_{j+1}^A \ge T_{k+1}^D$, then $y_{k+1,j+1}^{hlb} = y_{k+1,j+1} = 0$, and the relation holds since from the inductive hypothesis on the main claim, $B_{k+1}^{hlb} \le B_{k+1}$ for $k < j$. There are four cases to consider when $T_{j+1}^A < T_{k+1}^D$, based on which term in the expression giving $x_{j,i}^{hlb}$ in Figure 3.10 yields the minimum (i.e., whether $x_{k+1,j+1}^{hlb}$ is equal to x_{k+1}^{hlb}, $L - y_{k,j+1}^{hlb}$, $b_{c(j+1)}T_{k+1,j+1} - y_{k,j+1}^{hlb}$, or $b_{c(j+1)}T_{k+1,k+1}$).

***Case 1*:** $x_{k+1,j+1}^{hlb} = x_{k+1}^{hlb}$

Since relation (A.1) holds for *k* from the inductive hypothesis,

$$\begin{aligned} B_{k+1}^{hlb} + L - y_{k+1,j+1}^{hlb} &= \left(B_k^{hlb} + x_{k+1}^{hlb}\right) + L - \left(y_{k,j+1}^{hlb} + x_{k+1,j+1}^{hlb}\right) = B_k^{hlb} + L - y_{k,j+1}^{hlb} \\ &\le B_k + L - y_{k,j+1} \le B_k + L - y_{k,j+1} + \left(x_{k+1} - x_{k+1,j+1}\right) = B_{k+1} + L - y_{k+1,j+1}, \end{aligned} \tag{A.4}$$

which establishes relation (A.1) for *k*+1 for this case.

***Case 2*:** $x^{hlb}_{k+1,j+1} = L - y^{hlb}_{k,j+1}$

Since from the inductive hypothesis on the main claim, $B^{hlb}_{k+1} \le B_{k+1}$ for $k < j$,

$$\begin{aligned} B^{hlb}_{k+1} + L - y^{hlb}_{k+1,j+1} &= B^{hlb}_{k+1} + L - \left(y^{hlb}_{k,j+1} + x^{hlb}_{k+1,j+1}\right) \\ &= B^{hlb}_{k+1} + L - \left(y^{hlb}_{k,j+1} + \left(L - y^{hlb}_{k,j+1}\right)\right) = B^{hlb}_{k+1} \le B_{k+1} \le B_{k+1} + L - y_{k+1,j+1}, \end{aligned} \tag{A.5}$$

establishing relation (A.1) for *k*+1 for this case.

***Case 3*:** $x^{hlb}_{k+1,j+1} = b_{c(j+1)} T_{k+1,j+1} - y^{hlb}_{k,j+1}$

From the inductive hypothesis on the main claim, $B^{hlb}_{k+1} \le B_{k+1}$ for $k < j$. Also, since $T_{k+1,j+1}$ is the time from the arrival of request *j*+1 until the deadline of request *k*+1, and $y_{k+1,j+1}$ is the total amount of data received by the request *j*+1 client from transmissions also received by at least one other client, with request indexed at most *k*+1, it must be that $y_{k+1,j+1} \le b_{c(j+1)} T_{k+1,j+1}$. Therefore,

$$\begin{aligned} B^{hlb}_{k+1} + L - y^{hlb}_{k+1,j+1} &= B^{hlb}_{k+1} + L - \left(y^{hlb}_{k,j+1} + x^{hlb}_{k+1,j+1}\right) \\ &= B^{hlb}_{k+1} + L - \left(y^{hlb}_{k,j+1} + \left(b_{c(j+1)} T_{k+1,j+1} - y^{hlb}_{k,j+1}\right)\right) = B^{hlb}_{k+1} + L - b_{c(j+1)} T_{k+1,j+1} \\ &\le B_{k+1} + L - b_{c(j+1)} T_{k+1,j+1} \le B_{k+1} + L - y_{k+1,j+1}, \end{aligned} \tag{A.6}$$

establishing relation (A.1) for *k*+1 for this case.

***Case 4*:** $x^{hlb}_{k+1,j+1} = b_{c(j+1)} T_{k+1,k+1}$

This case is divided into sub-cases depending on the other requests, if any, whose deadlines fall between the arrival time and the deadline of request *k*+1.

***Case 4.1*:** $x^{hlb}_{k+1,j+1} = b_{c(j+1)} T_{k+1,k+1}$, and there is no request *i* ($i \le k$) such that $T^{A}_{k+1} < T^{D}_{i} \le T^{D}_{k+1}$.

Since there are no clients with earlier request deadlines that are able to share the transmissions required for request *k*+1, $B_{k+1} = B_k + L$. From this fact together with $x^{hlb}_{k+1} \le L$, $y_{k+1,j+1} - y_{k,j+1} = x_{k+1,j+1} \le b_{c(j+1)} T_{k+1,k+1}$, and since relation (A.1) holds for *k* from the inductive hypothesis, it follows that

$$\begin{aligned} B^{hlb}_{k+1} + L - y^{hlb}_{k+1,j+1} &= \left(B^{hlb}_{k} + x^{hlb}_{k+1}\right) + L - \left(y^{hlb}_{k,j+1} + x^{hlb}_{k+1,j+1}\right) \\ &= \left(B^{hlb}_{k} + L - y^{hlb}_{k,j+1}\right) + \left(x^{hlb}_{k+1} - b_{c(j+1)} T_{k+1,k+1}\right) \\ &\le \left(B_{k} + L - y_{k,j+1}\right) + \left(L - b_{c(j+1)} T_{k+1,k+1}\right) \\ &= \left(B_{k} + L\right) + L - \left(y_{k,j+1} + b_{c(j+1)} T_{k+1,k+1}\right) \le B_{k+1} + L - y_{k+1,j+1}, \end{aligned} \tag{A.7}$$

which establishes relation (A.1) for *k*+1 for this case.

***Case 4.2*:** $x^{hlb}_{k+1,j+1} = b_{c(j+1)} T_{k+1,k+1}$, and there is at least one request *i* ($i \le k$) such that $T^{A}_{k+1} < T^{D}_{i} \le T^{D}_{k+1}$ and such that $x^{hlb}_{i,j+1} = L - y^{hlb}_{i-1,j+1}$.

This case cannot occur since $x^{hlb}_{i,j+1} = L - y^{hlb}_{i-1,j+1}$ would imply that $x^{hlb}_{k+1,j+1} = 0$, in contradiction to the assumption that $x^{hlb}_{k+1,j+1} = b_{c(j+1)} T_{k+1,k+1}$.

Case 4.3: $x^{hlb}_{k+1,j+1} = b_{c(j+1)}T_{k+1,k+1}$, and there is at least one request i ($i \le k$) such that $T^A_{k+1} < T^D_i \le T^D_{k+1}$ and such that $x^{hlb}_{i,j+1} = b_{c(j+1)}T_{i,j+1} - y^{hlb}_{i-1,j+1}$.

Relationship $x^{hlb}_{k+1,j+1} = b_{c(j+1)}T_{k+1,k+1}$ and $y^{hlb}_{k,j+1} \ge y^{hlb}_{i,j+1}$ implies that $y^{hlb}_{k+1,j+1} = y^{hlb}_{k,j+1} + x^{hlb}_{k+1,j+1} \ge y^{hlb}_{i,j+1} + b_{c(j+1)}T_{k+1,k+1}$. Since $x^{hlb}_{i,j+1} = b_{c(j+1)}T_{i,j+1} - y^{hlb}_{i-1,j+1}$, and therefore $y^{hlb}_{i,j+1} = y^{hlb}_{i-1,j+1} + x^{hlb}_{i,j+1} = b_{c(j+1)}T_{i,j+1}$, this yields $y^{hlb}_{k+1,j+1} \ge b_{c(j+1)}T_{i,j+1} + b_{c(j+1)}T_{k+1,k+1}$. Using $T_{k+1,k+1} > T_{k+1,j+1} - T_{i,j+1}$, and the fact that $b_{c(j+1)}T_{k+1,j+1} \ge y_{k+1,j+1}$, this implies that $y^{hlb}_{k+1,j+1} > y_{k+1,j+1}$. Together with the inductive hypothesis on the main claim, $B^{hlb}_{k+1} \le B_{k+1}$ for $k < j$, this yields

$$B^{hlb}_{k+1} + L - y^{hlb}_{k+1,j+1} \le B_{k+1} + L - y^{hlb}_{k+1,i+1} < B_{k+1} + L - y_{k+1,j+1}, \qquad \text{(A.8)}$$

which establishes relation (A.1) for k+1 for this case.

Case 4.4: $x^{hlb}_{k+1,j+1} = b_{c(j+1)}T_{k+1,k+1}$, and all requests i ($i \le k$) such that $T^A_{k+1} < T^D_i \le T^D_{k+1}$ (of which there is at least one), are such that $x^{hlb}_{i,j+1} = x^{hlb}_i$.

Let $n > 0$ denote the number of such requests, indexed k+1-n through k. Given that $x^{hlb}_{i,j+1} = x^{hlb}_i$ for each such request i,

$$\begin{aligned} B^{hlb}_{k+1} + L - y^{hlb}_{k+1,j+1} &= \left(B^{hlb}_{k-n} + \sum_{i=k+1-n}^{k+1} x^{hlb}_i \right) + L - \left(y^{hlb}_{k-n,j+1} + \sum_{i=k+1-n}^{k+1} x^{hlb}_{i,j+1} \right) \\ &= B^{hlb}_{k-n} + L - y^{hlb}_{k-n,j+1} + \left(x^{hlb}_{k+1} - x^{hlb}_{k+1,j+1} \right). \end{aligned} \qquad \text{(A.9)}$$

Since relation (A.1) holds for k-n from the inductive hypothesis, this implies

$$B^{hlb}_{k+1} + L - y^{hlb}_{k+1,j+1} \le B_{k-n} + L - y_{k-n,j+1} + \left(x^{hlb}_{k+1} - x^{hlb}_{k+1,j+1} \right). \qquad \text{(A.10)}$$

Using $x^{hlb}_{k+1,j+1} = b_{c(j+1)}T_{k+1,k+1}$ and the fact that $x^{hlb}_{k+1} \le L$ yields

$$B^{hlb}_{k+1} + L - y^{hlb}_{k+1,j+1} \le B_{k-n} + L - y_{k-n,j+1} + \left(L - b_{c(j+1)}T_{k+1,k+1} \right). \qquad \text{(A.11)}$$

Consider now the total amount of data that the request j+1 client receives from transmissions also received by at least one of the clients with requests indexed k+1-n through k+1, but not received by any client with an earlier request deadline, i.e., $\sum_{i=k+1-n}^{k+1} x_{i,j+1}$. The portion of this data received after the arrival of request k+1 is upper bounded by $b_{c(j+1)}T_{k+1,k+1}$. The portion of this data received prior to the arrival of request k+1 is upper bounded by $\sum_{i=k+1-n}^{k+1} x_i - L$, since $\sum_{i=k+1-n}^{k+1} x_i$ gives the amount of data received by the clients with requests indexed k+1-n through k+1, from transmissions not received by any client with an earlier request deadline, and at least L of this data must be transmitted after the arrival of request k+1 so as to serve this request. (Note that all of the data received by the request k+1 client, must be from transmissions not received by any client with a request deadline earlier than that of request k+1-n, since such deadlines occur prior to the arrival of request k+1.) Thus, $\sum_{i=k+1-n}^{k+1} x_{i,j+1} \le b_{c(j+1)}T_{k+1,k+1} + \sum_{i=k+1-n}^{k+1} x_i - L$, or $L - b_{c(j+1)}T_{k+1,k+1} \le \sum_{i=k+1-n}^{k+1} x_i - \sum_{i=k+1-n}^{k+1} x_{i,j+1}$, yielding, when applied with the previous relation,

$$B_{k+1}^{hlb} + L - y_{k+1,j+1}^{hlb} \le B_{k-n} + L - y_{k-n,j+1} + \left(\sum_{i=k+1-n}^{k+1} x_i - \sum_{i=k+1-n}^{k+1} x_{i,j+1} \right) = B_{k+1} + L - y_{k+1,j+1}, \qquad \text{(A.12)}$$

and establishing relation (A.1) for k+1 for this case.

***Case 4.5*:** $x_{k+1,j+1}^{hlb} = b_{c(j+1)} T_{k+1,k+1}$, and all requests i ($i \le k$) such that $T_{k+1}^A < T_i^D \le T_{k+1}^D$ are such that either $x_{i,j+1}^{hlb} = b_{c(j+1)} T_{i,i}$ or $x_{i,j+1}^{hlb} = x_i^{hlb}$, with at least one having $x_{i,j+1}^{hlb} = b_{c(j+1)} T_{i,i}$.

Define two requests indexed p, q (p < q, and thus $T_p^D \le T_q^D$) as overlapping if $T_q^A < T_p^D$. Define "(in)directly overlapping" as the transitive closure of the overlapping relation. Let U denote the set of requests that are (in)directly overlapping with request j+1 when considering only request j+1 and those requests i such that $i \le k+1$ and $x_{i,j+1}^{hlb} = b_{c(j+1)} T_{i,i}$. Note that $|U| \ge 2$, since by the assumptions of this case, request k+1 is in U as is at least one other request. Let the index of the request in U with the earliest arrival time be denoted by e. Note that if $T_e^A \le T_{j+1}^A$, then, since $B_{k+1}^{hlb} \le B_{k+1}$ for $k < j$ from the inductive hypothesis on the main claim,

$$\begin{aligned} B_{k+1}^{hlb} + L - y_{k+1,j+1}^{hlb} &\le B_{k+1}^{hlb} + L - b_{c(j+1)} T_{k+1,j+1} \\ &\le B_{k+1} + L - b_{c(j+1)} T_{k+1,i+1} \le B_{k+1} + L - y_{k+1,j+1}, \end{aligned} \qquad \text{(A.13)}$$

which would establish relation (A.1) for k+1 for this case. Assume in the following that $T_e^A > T_{j+1}^A$.

Let V denote the set of requests i ($i \le k$) such that $T_e^A < T_i^D \le T_{k+1}^D$ and such that $x_{i,j+1}^{hlb} \ne b_{c(j+1)} T_{i,i}$. No request $i \in V$ can have $x_{i,j+1}^{hlb} = L - y_{i-1,j+1}^{hlb}$, by the same reasoning as used for case 4.2 above. Also, if for at least one request $i \in V$, $x_{i,j+1}^{hlb} = b_{c(j+1)} T_{i,j+1} - y_{i-1,j+1}^{hlb}$, then relation (A.1) is established for k+1 for this case using similar reasoning as used for case 4.3 above, i.e., from $y_{i,j+1}^{hlb} = y_{i-1,j+1}^{hlb} + x_{i,j+1}^{hlb} = b_{c(j+1)} T_{i,j+1}$ and $y_{k+1,j+1}^{hlb} > b_{c(j+1)} T_{k+1,j+1} \ge y_{k+1,j+1}$. Thus, in the following, assume that $x_{i,j+1}^{hlb} = x_i^{hlb}$ for each request $i \in V$.

Let B_U denote the total amount of data in the transmissions received by one or more of the set U clients. Note that these transmissions would be sufficient for serving a shorter request stream including *only* the requests in the set U. Therefore, from the inductive hypothesis on the main claim, B_U is lower bounded by the total amount of transmitted data that would be computed by the (more general, with the ε_j) heterogeneous lower bound algorithm, when applied to this reduced request stream. Denote the values computed for the reduced request stream, and the ε_j values used in this computation, with the superscript "*". It is possible to choose the ε_j^* values such that for each request in the reduced stream, i.e., each request $i \in U$, $x_i^{hlb*} = x_i^{hlb}$. To see this, note that for the request $i_1 \in U$ with the earliest deadline (and thus the first request in the reduced request stream), $\varepsilon_{i_1}^*$ can be chosen as $L - x_{i_1}^{hlb}$. For the request $i_2 \in U$ with the next earliest deadline, note that $y_{i_1,i_2}^{hlb} \ge y_{i_1,i_2}^{hlb*}$, since the presence of requests i in the full request stream with deadlines prior to that of i_1, and the resulting nonnegative

x_{i,i_2}^{hlb} values, cannot decrease y_{i_1,i_2}^{hlb}. Similarly, requests i intermediate between i_1 and i_2 in the full request stream contribute nonnegative x_{i,i_2}^{hlb} values, and thus $y_{i_2-1,i_2}^{hlb} \geq y_{i_1,i_2}^{hlb*}$, implying that $\varepsilon_{i_2}^{*}$ can be chosen as $y_{i_2-1,i_2}^{hlb} - y_{i_1,i_2}^{hlb*} + \varepsilon_{i_2}$. Similarly for the other requests in *U*; in general, the $\varepsilon_{i_l}^{*}$ values can be chosen in order of request deadline, such that $\varepsilon_{i_l}^{*} = y_{i_l-1,i_l}^{hlb} - y_{i_{l-1},i_l}^{hlb*} + \varepsilon_{i_l}$. Thus, $\sum_{i\in U} x_i^{hlb} = \sum_{i\in U} x_i^{hlb*} \leq B_U$.

Let $m \geq 2$ denote $|U|+|V|$. From $x_{i,j+1}^{hlb} = x_i^{hlb}$ for each request $i \in V$, the following holds

$$B_{k+1}^{hlb} + L - y_{k+1,j+1}^{hlb} = \left(B_{k+1-m}^{hlb} + \sum_{i\in U} x_i^{hlb} + \sum_{i\in V} x_i^{hlb} \right) + L - \left(y_{k+1-m,j+1}^{hlb} + \sum_{i\in U} x_{i,j+1}^{hlb} + \sum_{i\in V} x_{i,j+1}^{hlb} \right)$$

$$= B_{k+1-m}^{hlb} + L - y_{k+1-m,j+1}^{hlb} + \left(\sum_{i\in U} x_i^{hlb} - \sum_{i\in U} x_{i,j+1}^{hlb} \right), \qquad \text{(A.14)}$$

which implies, since relation (A.1) holds for *k*+1-*m* from the inductive hypothesis,

$$B_{k+1}^{hlb} + L - y_{k+1,j+1}^{hlb} \leq B_{k+1-m} + L - y_{k+1-m,j+1} + \left(\sum_{i\in U} x_i^{hlb} - \sum_{i\in U} x_{i,j+1}^{hlb} \right). \qquad \text{(A.15)}$$

Since $x_{i,j+1}^{hlb} = b_{c(j+1)} T_{i,i}$ for each request $i \in U$, $b_{c(j+1)} T_{k+1,e} \leq \sum_{i\in U} x_{i,j+1}^{hlb}$. Together with $\sum_{i\in U} x_i^{hlb} \leq B_U$, this yields

$$B_{k+1}^{hlb} + L - y_{k+1,j+1}^{hlb} \leq B_{k+1-m} + L - y_{k+1-m,j+1} + \left(B_U - b_{c(j+1)} T_{k+1,e} \right). \qquad \text{(A.16)}$$

Consider now the total amount of data that the request *j*+1 client receives from transmissions also received by at least one of the clients with requests indexed *k*+2-*m* through *k*+1, but not received by a client with an earlier request deadline, i.e., $\sum_{i=k+2-m}^{k+1} x_{i,j+1}$. The portion of this data received after the arrival of request *e* is upper bounded by $b_{c(j+1)} T_{k+1,e}$. The portion of this data received prior to the arrival of request *e* is upper bounded by $\sum_{i=k+2-m}^{k+1} x_i - B_U$, since $\sum_{i=k+2-m}^{k+1} x_i$ gives the amount of data received by the clients with requests indexed *k*+2-*m* through *k*+1, from transmissions not received by a client with an earlier request deadline, and at least B_U of this data is transmitted after the arrival of request *e*, as it is received by one or more set *U* clients. (Note that all of the data received by set *U* clients, must be from transmissions not received by any client with a request deadline earlier than that of request *k*+2-*m*, since such deadlines occur prior to the arrival of any of the set *U* clients.) Thus, $\sum_{i=k+2-m}^{k+1} x_{i,j+1} \leq b_{c(j+1)} T_{k+1,e} + \sum_{i=k+2-m}^{k+1} x_i - B_U$, or $B_U - b_{c(j+1)} T_{k+1,e} \leq \sum_{i=k+2-m}^{k+1} x_i - \sum_{i=k+2-m}^{k+1} x_{i,j+1}$, yielding, when applied with the previous relation,

$$B_{k+1}^{hlb} + L - y_{k+1,j+1}^{hlb} \leq B_{k+1-m} + L - y_{k+1-m,j+1} + \left(\sum_{i=k+2-m}^{k+1} x_i - \sum_{i=k+2-m}^{k+1} x_{i,j+1} \right) = B_{k+1} + L - y_{k+1,j+1}, \qquad \text{(A.17)}$$

which establishes relation (A.1) for *k*+1 for this case.

As the above cases are mutually exhaustive, relation (A.1) is established, and thus also the main claim.

Appendix B

Asymptotic Analysis of Dynamic vs. Static Replica Selection

Assuming that the probability that a request could receive service with more than one other request is negligibly small, the optimal static policy is for each request to be served by the local replica. Under this assumption, for the batched service model, a request arrival occurring at time *t* only causes a service initiation (at time *t* + *D*) if no other request arrival has occurred within (t – *D*, *t*]. Using this observation, the total service delivery cost in a system with identical client group request rates (i.e., $\lambda_i = \lambda/N$) can be calculated as

$$\lambda e^{-(\lambda/N)D} L \approx \lambda L\left(1-\frac{\lambda}{N}D\right), \tag{B.1}$$

where Taylor expansions have been used for the final expression. For the fountain service model, on average a request is able to share half its service with a local request that arrives within *D* of itself. Using this observation, the corresponding cost under the fountain service model can be approximated as

$$\lambda\left(e^{-(\lambda/N)D} L + (1-e^{-(\lambda/N)D})L/2\right) \approx \lambda L\left(1-\frac{\lambda}{2N}D\right). \tag{B.2}$$

As described in Section 4.2.4, the optimal dynamic policy is for each request to be served by the local replica if no previous request is waiting for service (in the case of batched service) or receiving service (in the case of fountain service) at the time of arrival of the request. In the rare event that there is such a previous request, the cost is minimized if the new request shares its service (all in the case of batched service, or whatever service remains for the previous request in the case of fountain service) with this previous request (and, in the case of fountain service, receives the remaining portion of its service locally).

Similar to the above analysis, the total service delivery cost using the batched and fountain service model can be calculated as

$$\lambda\left(e^{-\lambda D} L + (1-e^{-(\lambda/N)D})\frac{N-1}{N}cL\right) \approx \lambda L\left(1-\lambda\left(1-\frac{N-1}{N}c\right)D\right), \tag{B.3}$$

and

$$\lambda\left(e^{-\lambda D} L + (1-e^{-(\lambda/N)D})\left[\frac{N-1}{N}\left(c\frac{L}{2}+\frac{L}{2}\right)+\frac{1}{N}\frac{L}{2}\right]\right) \approx \lambda L\left(1-\lambda\left(1-\frac{N-1}{N}c\right)\frac{D}{2}\right), \tag{B.4}$$

respectively. With equation B.1 to B.4 being linear equations, *D* can easily be solved for. With the batched service model the maximum client delay, using the optimal static and the optimal dynamic policy, respectively, can be calculated as

$$D_{static} \approx \frac{N}{\lambda}\left(1-\frac{C}{\lambda L}\right);\ D_{dynamic} \approx \frac{1}{\lambda(1-c(N-1)/N)}\left(1-\frac{C}{\lambda L}\right). \tag{B.5}$$

The corresponding delays, using the fountain service model, can be calculated as

$$D_{static} \approx \frac{2N}{\lambda}\left(1-\frac{C}{\lambda L}\right);\ D_{dynamic} \approx \frac{2}{\lambda(1-c(N-1)/N)}\left(1-\frac{C}{\lambda L}\right). \tag{B.6}$$

For both service models, using these asymptotic limits and simple algebra, the asymptotic delay differences reduce to $(N–1)(1–c) \times 100\%$.

Appendix C

Analysis of Heterogeneous Batched At-arrival Policy

This appendix outlines an analysis of the optimal at-arrival replica selection policy for the fountain service model, with policy and the threshold parameters T_i, T_i', and T_i'' defined in Section 4.3.1.2.

As in the case of homogenous client groups, the analysis proceeds by considering the state of the system at an arbitrary time t under the operation of an optimal at-arrival policy. Consider first a replica and client group i other than replica/group 1. If there has been at least one request arrival from client group i in the time interval $[t–L/r, t–T_i]$, replica i will be dispensing service at time t. If there have been no requests from client group i but at least one request from some other client group j in the time interval $[t – L/r, t–T_i]$ (and thus replica j is dispensing service at time t), all requests that arrive from client group i in the time interval $[t–T_i, t]$ will be receiving service from a remote replica at time t. If there have been no requests from any client group in the time interval $[t–L/r, t–T_i]$ but at least one request in the interval $[t–T_i, t]$, and the first such request was from a client group other than group i, all requests that arrive from client group i in the time interval $[t–T_i, t]$ will be receiving service from a remote replica at time t (note that the expected number of such requests must be conditioned on there being at least one request arrival, with the first such being from other than group i). If there have been no requests from any client group in the time interval $[t–L/r, t–T_i]$ but at least one request in the interval $[t–T_i, t–T_i']$, or no requests from any client group in the time interval $[t–L/r, t–T_i'']$ but at least one request in the interval $[t–T_i'', t]$, and the first such request was from client group i, then replica i will be dispensing service at time t. (The above corresponds to the first four terms within the first set of curly brackets of equation (C.1).)

The remaining case that has non-zero expected cost is where there have been no requests from any client group in the time interval $[t–L/r, t–T_i']$ but at least one request in the interval $[t–T_i', t–T_i'']$, and the first such request was from client group i. In this case, all requests that arrive from client group i in the time intervals $[t–T_i', t–T_i'']$ and $[t–T_i'', t]$ will be receiving service from replica 1 at time t. Referring to equation (C.1), terms six and seven in the first set of curly brackets correspond to the remote access cost of these requests, while terms eight and nine refer to the service cost of replica 1, initiated by these replica i requests (while compensating for the fact that the analysis for replica 1 does not take these request into consideration), at the times during the intervals $[t–T_i', t–\max[T_1, T_i'']]$ and $[t–\max[T_1, T_i''], t–T_i'']$, respectively.

The analysis for replica and client group 1 follows a similar approach. In the resulting analytic expression for the total service delivery cost, as shown below, the terms for replica and client group 1 (within the second set of curly braces), neglect the fact that requests from other than client group 1 can cause replica 1 to be scheduled; this is compensated for with the last two terms for each replica/client group i (within the first set of curly braces):

$$
\begin{aligned}
& r\sum_{i=2}^{N}\Big\{\left(1-e^{-\lambda_i(L/r-T_i)}\right)+\left(e^{-\lambda_i(L/r-T_i)}-e^{-\lambda(L/r-T_i)}\right)c\lambda_i T_i+e^{-\lambda(L/r-T_i)}(1-f_i)c\left(\lambda T_i-\left(1-e^{-\lambda T_i}\right)\right)f_i \\
&\quad +\left(\left(e^{-\lambda(L/r-T_i)}-e^{-\lambda(L/r-T_i')}\right)+\left(e^{-\lambda(L/r-T_i'')}-e^{-\lambda L/r}\right)\right)f_i \\
&\quad +e^{-\lambda(L/r-T_i')}f_i c\left(\lambda_i(T_i'-T_i'')+\left(1-e^{-\lambda(T_i'-T_i'')}\right)(1-f_i)\right)+\left(e^{-\lambda(L/r-T_i')}-e^{-\lambda(L/r-T_i'')}\right)f_i c\lambda_i T_i'' \\
&\quad +e^{-\lambda(L/r-T_i')}e^{-\lambda_1(T_i'-T_1)}\left(1-e^{-(\lambda-\lambda_1)(T_i'-\max[T_1,T_i''])}\right)(\lambda_i/(\lambda-\lambda_1))(1-c\lambda_1 T_1) \\
&\quad +\left(e^{-\lambda(L/r-\max[T_1,T_i''])}-e^{-\lambda(L/r-T_i'')}\right)f_i\left(1-c\left(\lambda_1\max[T_1,T_i'']+\left(\lambda\max[0,T_1-T_i'']/\left(1-e^{-\lambda\max[0,T_1-T_i'']}\right)-1\right)f_1\right)\right)\Big\} \\
&\quad +r\Big\{\left(1-e^{-\lambda_1(L/r-T_1)}\right)+\left(e^{-\lambda_1(L/r-T_1)}-e^{-\lambda(L/r-T_1)}\right)c\lambda_1 T_1+\left(e^{-\lambda(L/r-T_1)}-e^{-\lambda L/r}\right)f_1 \\
&\quad +e^{-\lambda(L/r-T_1)}(1-f_1)c\left(\lambda T_1-\left(1-e^{-\lambda T_1}\right)\right)f_1\Big\}, \qquad \text{(C.1)}
\end{aligned}
$$

where f_i corresponds to the fraction of requests that are from client group i. The correctness of this analysis has been checked using simulation.