**Scalable On-demand Streaming of Stored Complex Multimedia**

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

Yanping Zhao

# Permission To Use

In presenting this thesis in partial fulfillment of the requirements for a Post-graduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science
University of Saskatchewan
Saskatoon, Saskatchewan, Canada
S7N 5A9

# Abstract

*Previous research has developed a number of efficient protocols for streaming popular multimedia files on-demand to potentially large numbers of concurrent clients. These protocols can achieve server bandwidth usage that grows much slower than linearly with the file request rate, and with the inverse of client start-up delay. This thesis makes the following three main contributions to the design and performance evaluation of such protocols.*

*The first contribution is an investigation of the network bandwidth requirements for scalable on-demand streaming. The results suggest that the minimum required network bandwidth for scalable on-demand streaming typically scales as* $K/\ln(K)$ *as the number of client sites* $K$ *increases for fixed request rate per client site, and as* $\ln\left(\frac{N}{ND+1}\right)$ *as the total file request rate* $N$ *increases or client start-up delay* $D$ *decreases, for a fixed number of sites. Multicast delivery trees configured to minimize network bandwidth usage rather than latency are found to only modestly reduce the minimum required network bandwidth. Furthermore, it is possible to achieve close to the minimum possible network and server bandwidth usage simultaneously with practical scalable delivery protocols.*

*Second, the thesis addresses the problem of scalable on-demand streaming of a more complex type of media than is typically considered, namely variable bit rate (VBR) media. A lower bound on the minimum required server bandwidth for scalable on-demand streaming of VBR media is derived. The lower bound analysis motivates the design of a new immediate service protocol termed VBR bandwidth skimming (VBRBS) that uses constant bit rate streaming, when sufficient client storage space is available, yet fruitfully exploits the knowledge of a VBR profile.*

*Finally, the thesis proposes non-linear media containing parallel sequences of data frames, among which clients can dynamically select at designated branch points, and investigates the design and performance issues in scalable on-demand streaming of such media. Lower bounds on the minimum required server bandwidth for various non-linear media scalable on-demand streaming approaches are derived, practical non-linear media scalable delivery protocols are developed, and, as a proof-of-concept, a simple scalable delivery protocol is implemented in a non-linear media streaming prototype system.*

# Acknowledgements

I would like to express my sincere appreciation to my supervisor Dr. Derek Eager for his tremendous help to my dissertation research. Without his encouragement, patience, guidance, and knowledge, this dissertation would be impossible. Dr. Eager has put enormous amount of effort to train me with knowledge, skills, and ethics of doing research. I believe that what I have learned from him will continue benefiting my academic career.

I want to give my special thanks to Dr. Rick Bunt and Dr. Dwight Makaroff, the two professors in our research group and in my Ph.D. program committee. Dr. Bunt gave me many insightful comments and suggestions on my thesis. He has been very considerate and supportive to my study and my family. Dr. Makaroff gave me many valuable advice and feedback to improve the quality of my thesis. I really enjoyed discussions with him.

I would like to thank Brian Gallaway and Greg Oster, our system administrators, for their professional and timely technical support. I want to thank Brian Gallaway, Peter O'Donovan, and Jeremy Parker for their contributions and assistance to the prototyping component of my thesis.

The office staff members of my department have been wonderful. I especially want to thank Jan Thompson, our graduate correspondent, for her love and caring of students. Jan has been a good friend to me and has made my graduate study very special.

I would like to thank Dr. Mary Vernon for providing valuable feedback to my research and for assisting with paper writing. Collaborating with her is a very pleasant experience.

I must acknowledge my other committee members, Dr. Jonny Wong (external), Dr. Raj Srinivasan (cognate), and Dr. Mark Keil for their helpful comments on my thesis.

I am very grateful to Dr. Anirban Manhanti and Dr. Carey Williamson for all the help they have given to me.

Finally, my deepest appreciation to my parents and my husband. My parents gave me unconditional help in my most difficult times. My husband has always been my best friend and research partner.

# Table of Contents

**Permission To Use** i

**Abstract** ii

**Acknowledgements** iii

**Table of Contents** v

**List of Tables** ix

**List of Figures** x

**List of Acronyms** xiii

**1 Introduction** 1

1.1 Streaming of Stored Multimedia . . . . . 2

1.1.1 Overview . . . . . 2

1.1.2 Scalable Streaming of Stored Multimedia . . . . . 5

1.1.2.1 Content Replication and Caching . . . . . 5

1.1.2.2 Scalable Delivery Protocols . . . . . 6

1.1.3 Complex Multimedia . . . . . 8

1.2 Contributions of the Thesis . . . . . 10

1.2.1 Network Bandwidth Bounds . . . . . 11

1.2.2 Scalable On-demand Streaming of VBR Media . . . . . 12

1.2.3 Scalable On-demand Streaming of Non-linear Media . . . . . 13

1.3 Thesis Organization . . . . . 14

**2 Related Research** 15

2.1 Continuous Media Overview . . . . . 16

2.2 Multicast Service Model . . . 19
2.2.1 Multicast Distribution Trees . . . 20
2.2.2 Source-specific Multicast . . . 22
2.2.3 Application Layer Multicast . . . 23
2.3 Scalable Delivery Protocols . . . 23
2.3.1 Periodic Broadcast Protocols . . . 25
2.3.2 Hierarchical Stream Merging . . . 28
2.3.3 Bandwidth Skimming . . . 29
2.4 Bandwidth Scalability Bounds . . . 33

**3 Network Bandwidth for Scalable On-demand Streaming 37**
3.1 Multicast Delivery Network Bandwidth . . . 38
3.2 Analysis for Minimum Network Bandwidth . . . 40
3.2.1 Performance Metric and Analysis Assumptions . . . 40
3.2.2 Shared Link with Fan-out $K$ Topology . . . 41
3.2.3 Daisy-chain Topology . . . 45
3.2.4 Balanced Tree Topology . . . 48
3.3 Multicast Delivery in Large Networks . . . 52
3.3.1 Network Topology Generators . . . 53
3.3.2 Tree Construction Algorithms . . . 54
3.3.3 Network Bandwidth Results . . . 56
3.3.3.1 Comparison of Tree Construction Algorithms . . . 57
3.3.3.2 Scaling of Minimum Network Bandwidth . . . 59
3.4 Network Bandwidth of Hierarchical Stream Merging . . . 60
3.4.1 Network-naïve Hierarchical Stream Merging . . . 61
3.4.2 Network-aware Hierarchical Stream Merging . . . 63
3.5 Network Bandwidth of Periodic Broadcast . . . 66
3.5.1 OPB/tou . . . 66
3.5.2 Network Bandwidth of OPB/tou . . . 68
3.6 Summary . . . 70

**4 Scalable On-demand Streaming of VBR Media 72**

4.1 Work-ahead Smoothing . . . 74

4.2 VBR Trace Characteristics . . . 77

4.3 Impact of VBR on Server Bandwidth . . . 80

4.3.1 Server Bandwidth for Streaming VBR Media . . . 80

4.3.2 Impact of Work-ahead Smoothing on VBR Delivery . . . 82

4.4 VBR Bandwidth Skimming . . . 83

4.4.1 Protocol Description . . . 84

4.4.2 Determining Merge Points . . . 86

4.4.3 Determining Commit Points . . . 89

4.4.4 Accommodating Limited Client Storage . . . 91

4.5 Performance Evaluation of VBRBS . . . 93

4.5.1 Impact of File Request Rate . . . 93

4.5.2 Impact of Client Receive Bandwidth . . . 95

4.5.3 Impact of Client Buffer Capacity . . . 96

4.6 Summary . . . 96

**5 Scalable On-demand Streaming of Non-linear Media 98**

5.1 Non-linear Media Models . . . 100

5.1.1 Non-linear Media Structures . . . 100

5.1.2 Client Branch Selections . . . 101

5.1.3 A Sample Non-linear Media Tree . . . 102

5.1.4 Server Knowledge of Client Path Selection . . . 103

5.2 Analysis for Minimum Server Bandwidth . . . 104

5.2.1 Potential for Scalable Delivery . . . 105

5.2.1.1 Minimum Required Server Bandwidth . . . 106

5.2.1.2 Client Data Overhead . . . 110

5.2.2 Restricted Snoop-ahead Approaches . . . 113

5.2.2.1 Distance-based Restricted Snoop-ahead . . . 113

5.2.2.2 Client Path Prediction Approaches . . . 116

5.2.2.3 Policy Comparisons . . . 120
5.3 Scalable Delivery Protocols . . . 124
5.3.1 Hierarchical Stream Merging . . . 124
5.3.2 Optimized Periodic Broadcast . . . 134
5.3.2.1 OPB Known Path (OPB-KP) . . . 135
5.3.2.2 OPB Unknown Path (OPB-UP) . . . 139
5.3.3 Performance Comparisons . . . 144
5.4 Prototype Implementation . . . 144
5.4.1 The SWORD Prototype . . . 145
5.4.2 Non-linear Media Streaming in SWORD . . . 151
5.4.2.1 Constructing Non-linear Media Files . . . 151
5.4.2.2 SWORD Client Component Design . . . 153
5.4.2.3 SWORD Server Component Design . . . 156
5.4.3 Experiments with Non-linear Media Streaming . . . 159
5.5 Summary . . . 171

**6 Conclusions** **172**
6.1 Thesis Summary . . . 173
6.2 Thesis Contributions . . . 175
6.3 Future Work . . . 175

**References** **178**

# List of Tables


2.1 Notation for Server Bandwidth Bounds . . . . . 34

3.1 Notation for Minimum Network Bandwidth Analysis . . . . . 41
3.2 Scaling of Network Bandwidth for Multicast . . . . . 52
3.3 Experimental Parameters . . . . . 57
3.4 Notation for OPB/tou . . . . . 67

4.1 Object Characteristics . . . . . 79

5.1 Notation for Non-linear Media Scalable Delivery Approaches . . . . . 105
5.2 Notation for OPB . . . . . 135
5.3 Notation for OPB-UP . . . . . 142
5.4 Events in Experiment 1 . . . . . 164
5.5 Events in Experiment 2 . . . . . 167
5.6 Events in Experiment 3 . . . . . 170

# List of Figures


2.1 Optimized Periodic Broadcast Protocol . . . . . 27
2.2 Hierarchical Stream Merging . . . . . 30
2.3 Latest Patch . . . . . 31
2.4 Partition . . . . . 33
2.5 Server Bandwidth Efficiency of Hierarchical Stream Merging . . . . . 35

3.1 Shared Link with Fan-out $K$ Topology . . . . . 42
3.2 Tightness of Bounds for Shared Link with Fan-out $K$ Topology . . . 45
3.3 Daisy-chain Topology . . . . . 46
3.4 Tightness of Bounds for Daisy-chain Topology . . . . . 48
3.5 Balanced Tree Topology . . . . . 49
3.6 Tightness of Bounds for Balanced Tree Topology . . . . . 52
3.7 Network Bandwidth for Alternative Delivery Trees . . . . . 58
3.8 Scaling of Minimum Network Bandwidth . . . . . 60
3.9 Bandwidth Efficiencies of Hierarchical Stream Merging . . . . . 63
3.10 Fan-out Topology With No Shared Links . . . . . 64
3.11 Network-aware vs. Network-naïve Hierarchical Stream Merging . . . . 65
3.12 Network-aware vs. Network-naïve Polices with Shared Link . . . . . 66
3.13 Server Bandwidth Efficiency of OPB/tou . . . . . 68
3.14 Network Bandwidth Usage of OPB/tou . . . . . 69
3.15 Network Bandwidth Efficiency of OPB/tou . . . . . 71

4.1 Synthetic Object Bit Rate Profiles . . . . . 77
4.2 Composite Object Bit Rate Profiles . . . . . 78
4.3 Video Clip Bit Rate Profiles . . . . . 78
4.4 Server Bandwidth for Unsmoothed VBR Files . . . . . 81
4.5 Server Bandwidth for Smoothed Synthetic Bit Rate Profiles . . . . . 83

4.6 Server Bandwidth for Smoothed Composite and Video Files . . . . . 84
4.7 VBR Bandwidth Skimming . . . . . 85
4.8 Server Bandwidth Used for Transmitting Redundant Data . . . . . 87
4.9 Algorithm for Generating a Merge Table . . . . . 88
4.10 Two Scenarios in the Algorithm of Figure 4.9 . . . . . 89
4.11 Server Bandwidth Usage for Varying Commit Point Placements . . . 91
4.12 Impact of File Request Rate . . . . . 94
4.13 Impact of Client Receive Bandwidth . . . . . 95
4.14 Impact of Client Buffer Capacity . . . . . 97

5.1 Tree Structure for Non-linear Media . . . . . 100
5.2 A Sample Non-linear Media Tree . . . . . 103
5.3 Server Bandwidth for Non-linear Media . . . . . 108
5.4 Impact of Tree Height . . . . . 109
5.5 Impact of Popularity Model . . . . . 110
5.6 Client Data Overhead for *Unrestricted Snoop-ahead* . . . . . 112
5.7 Server Bandwidth for Restricted Snoop-ahead Approaches . . . . . 121
5.8 Impact of Tree Height on Restricted Snoop-ahead Approaches . . . . 122
5.9 Client Overhead with Alternative Snoop-ahead Approaches . . . . . 123
5.10 Impact of Tree Height on Snoop-ahead Approaches . . . . . 123
5.11 Sensitivity to Skewness in Selection Probabilities . . . . . 125
5.12 An Illustration of the Inefficiency of *Path* . . . . . 129
5.13 Server Bandwidth Requirements of HSM Policies . . . . . 131
5.14 Server Bandwidth Requirements of HSM-POP . . . . . 133
5.15 Server Bandwidth Requirements of HSM-NEXT . . . . . 134
5.16 Broadcasting a Segment that Crosses a Branch Point . . . . . 136
5.17 OPB-KP Segment Partitioning for an Example Media Structure . . . 137
5.18 OPB-KP Channels for Structure of Fig. 5.17 . . . . . 137
5.19 Server Bandwidth Usage for a Cross-over Segment . . . . . 138
5.20 OPB-UP Segment Partitioning for an Example Media Structure . . . 140

5.21 OPB-UP Channels for Structure of Fig. 5.20 . . . 141
5.22 Algorithm for OPB-UP Segment Sizes . . . 143
5.23 Performance of Scalable Delivery Protocols . . . 145
5.24 Performance with Varying Height . . . 145
5.25 Impact of Start-up Delay . . . 146
5.26 SWORD Prototype Architecture . . . 147
5.27 SWORD Server/Client Hand-shaking Interaction . . . 148
5.28 HSM in SWORD . . . 149
5.29 An Example Non-linear Media File . . . 151
5.30 Transparency to Media Server and Media Client . . . 152
5.31 An ASF Movie File . . . 154
5.32 An ASF Data Packet . . . 155
5.33 Fields Changed When Modifying an ASF File for Convergence . . . 156
5.34 File Convergence at SWORD Client Component . . . 157
5.35 Experimental Environment and Set-up . . . 160
5.36 Non-linear Media File Used in Experiment 1 . . . 162
5.37 Experiment 1 . . . 163
5.38 Non-linear Media File Used in Experiment 2 . . . 165
5.39 Experiment 2 . . . 166
5.40 Non-linear Media File Used in Experiment 3 . . . 168
5.41 Experiment 3 . . . 169

# List of Major Acronyms

**AS** – Autonomous system

**CBR** – Constant bit rate

**GT-ITM** – Georgia tech internetwork topology models

**HSM** – Hierarchical stream merging

**HTTP** – Hypertext transfer protocol

**IP** – Internet protocol

**MPEG** – Motion pictures expert group

**OPB** – Optimized periodic broadcast

**STB** – Set-top box

**SWORD** – Scalable wide-area on-demand reliable digital streaming

**TCP** – Transport control protocol

**UDP** – User datagram protocol

**VBR** – Variable bit rate

**VBRBS** – VBR bandwidth skimming

**WWSTP** – Wisconsin Washington Saskatchewan transport protocol

# Chapter 1

# Introduction

Recent years have witnessed a blossoming of multimedia applications, such as video-on-demand, distance learning, movie previewing, teleconferencing, and live broadcast. These applications rely heavily on the *streaming* technique to deliver video and audio files to clients. Streaming allows clients to begin the playback of a file before its delivery from the media server is complete, rather than downloading the whole file onto the disk and then starting the playback.

Multimedia streaming applications roughly fall into three categories: on-demand streaming of stored multimedia (e.g., video-on-demand), live streaming (e.g., Internet radio stations), and real-time interactive streaming (e.g., on-line games and video conferencing). This thesis focuses mainly on design and performance issues in the on-demand streaming of stored video and audio, especially video files, over wide-area networks. Live and real-time interactive streaming are in some aspects similar to stored multimedia streaming, e.g., demanding continuous playback, but have their own characteristics and technical challenges, including stringent end-to-end delay constraints, which are out of the scope of the thesis.

On-demand streaming of stored multimedia requires the involvement of three components: the server, the network, and the clients. Their respective functions, behaviours, and properties are described in Section 1.1.1. To ensure quality streaming, many crucial performance issues need to be addressed, such as scalability, reliability, rate control, and congestion control. This thesis particularly concerns on-demand scalable streaming of stored audio and video files to a large number of geographically distributed clients using *scalable delivery protocols*, which are discussed in Section 1.1.2. Section 1.2 summarizes the contributions of the thesis to the de-

sign and performance study of scalable delivery protocols for complex multimedia objects, and Section 1.3 lays out the structure of the thesis.

## 1.1 Streaming of Stored Multimedia

### 1.1.1 Overview

A streaming system for distributing stored video/audio consists of three components: the server, the network, and the clients.

**Server**

The main functions of a streaming server are processing client requests, retrieving media files, and transmitting data into the network. To satisfy a client request, the server needs to use some CPU power to parse the request, some disk bandwidth to bring the requested data into in-memory buffers, and some network interface bandwidth to transmit data into the network. The resources along the access path necessary for a single media stream are collectively referred to as a *server channel*.

Since multimedia files, especially video files, normally are several minutes to several hours in length, a server channel, once allocated, may be occupied for a relatively long time. As streaming applications become more and more popular, the server may need to support hundreds of thousands of requests at the same time. Obviously, if the server dedicates a separate channel for each client request, the server channels will be quickly saturated. One approach to increasing server capacity is to build a server farm with hundreds or thousands of low cost desktop machines. This approach, however, is not the ultimate cure to the saturation problem since it does not address the potential bottleneck at the interface between the server egress router and the outside network, i.e., the server *network access bandwidth*. Simply provisioning more network access bandwidth to meet the scaling demand of the system may be costly. Also, as server network access bandwidth scales up, new applications that can quickly use the additional available bandwidth may emerge, making the access bandwidth still a constrained resource.

### Network

Once the requested file data is sent into the network, they are routed and transported to the client site where the request for the file was initiated. While the data are flowed through the network, some resources (e.g., processing power and buffer space) at each router and bandwidth on each link are consumed. These resources are collectively referred to as a *network channel*.

Currently, most streaming applications deliver streams using *unicast*, that is, each stream is delivered by a distinct network channel. If many users are accessing the same file during the same period of time, many copies of the same data may flow over the same links at the same time. Clearly, use of unicast not only wastes network resources, but may also saturate network channels when being used to stream multimedia files, which usually need to hold the channels for a relatively long time, from minutes to a couple of hours. Unicast similarly wastes server resources.

To address the above problems of unicast, *multicast* has been proposed to efficiently deliver data from one point to multiple points. Multicast can be implemented either inside the network (i.e., *IP multicast*) or at end hosts (i.e., *application layer multicast*). Although network layer multicast is very efficient in terms of network bandwidth usage, it suffers from many problems, such as router overhead, scalability, reliability, and security. One of the most important reasons that network layer multicast has so many troubles is that its service model is *many-to-many* in nature, that is, during a multicast session, anyone can send messages to all receivers and receive messages.

In many streaming applications, there is only a single server sending data to a large number of clients. Such applications motivate a new and simpler multicast service model called *source-specific multicast* (i.e., ssm) [18] that supports only *one-to-many* communications. Since there is only a single sender, group management, pricing, routing, and security issues in source-specific multicast can all be handled more easily and effectively than in traditional many-to-many multicast.

Application layer multicast [23, 24, 62, 90, 129] has been proposed as an alternative approach to providing one-to-many (or many-to-many) communication service. The basic idea is to let end hosts themselves, instead of network layer routers, copy and forward data to their downstream hosts. This idea is implemented by building a virtual network, called an *overlay*, across all participating end hosts. Each virtual link from one host to another in the overlay network is actually a unicast path. One-to-many multicast is implemented by constructing a multicast distribution tree from the server to all member nodes on top of the overlay. Data are then transmitted along the distribution tree, replicated and forwarded by the nodes at branch points, until received by every member node. Application layer multicast does not require network support for one-to-many communication. However, it has its own problems, for example, how an efficient overlay network can be created across all participating end hosts, and how the overlay network can be effectively maintained as network conditions change [96, 13, 127].

### Client

A client using a multimedia streaming system requires special software (e.g., Microsoft Media Player [80]) or specialized hardware (e.g., a set-top box (STB)) to play back the requested file. Since audio and video have high storage and bandwidth requirements, audio and video files are normally compressed when stored or delivered across a network. On the client side, the received file data must be decompressed before being rendered onto the screen. Video and audio are also delay-sensitive, that is, a particular piece of data, e.g., a video frame, has to be received and displayed by a particular time. After the deadline, it becomes useless. However, the current Internet only supports best-effort service that cannot guarantee an end-to-end delay bound, nor a bound on the variance of delays (i.e., jitter). On the client side, a start-up delay, possibly up to tens of seconds for streaming video, is usually introduced to accommodate delay-jitter, implying the need for a buffer to hold file data until it is played out.

### 1.1.2 Scalable Streaming of Stored Multimedia

Although several media-on-demand implementations have been carried out [17, 104], achieving scalability in multimedia streaming is still a crucial practical concern and an active research area. This section discusses two main approaches that have been proposed to address this issue, namely *content replication and caching* (Section 1.1.2.1) and *scalable delivery protocols* (Section 1.1.2.2). These two approaches are complementary to each other. However, the focus of this section will be on scalable delivery protocols since they are the focus of this thesis.

#### 1.1.2.1 Content Replication and Caching

This approach places copies of popular files close to end users through use of either proxy caches (as are widely used in the World Wide Web) or distributed servers at which the files are mirrored.

##### Proxy Caching

Proxy caches are commonly deployed to provide local storage of Web objects requested by clients in a particular organization or geographic area. The proxy intercepts all Web requests. If the object requested by a client is not in the proxy cache, the proxy sends the request to the respective server on behalf of the client. After receiving the requested data from the server, the proxy forwards it to the client, and at the same time makes a copy in its local storage so that subsequent requests for the same object can be served directly from the proxy. The use of a proxy not only reduces the average response time of client requests but also reduces the wide-area network traffic between the proxy and the servers.

Some previous research has considered specifically the application of proxy caching to distribute continuous media files. For example, one approach that has been proposed is to preload the prefixes (i.e., the first several minutes of a media file) of 10 to 20 of the most popular files into the proxy cache to reduce the start-up delay experienced by the clients requesting them in a video-on-demand system [103].

Other research has investigated the optimal placement of media file data across the server and the proxies and found that the optimal placement typically follows an all-or-nothing rule, that is, each file is either entirely stored in the proxies or not at all [5].

Since the capacity of a proxy cache is limited and since the size of a continuous media file, especially a video file, may be very large, the choice of cache replacement algorithm may be important to the effectiveness of the proxy. Traditional replacement algorithms, such as LRU and LFU, which have worked well with non-continuous media (bulk data) files, may not be suitable for continuous media files [28]. Designing efficient replacement algorithms for continuous media files is still an active research topic [6].

#### Content Replication

The basic idea of content replication is to deploy servers with copies of popular content throughout the Internet, forming a content delivery network (CDN). Each client request is directed by the CDN to the server that can satisfy the request at the minimum cost. Content distribution networks have been successfully built by companies such as Akamai [4] and Digital Island [33]. However, a few critical issues have not yet been answered completely, such as how content servers are optimally placed across networks, how client requests are best directed to the proper server, how server and network status information is maintained and updated, and how updates to replicated files are propagated to all servers in a timely fashion [93, 94, 66, 67, 118, 95, 111].

### 1.1.2.2 Scalable Delivery Protocols

The use of scalable delivery protocols is a complementary approach to content caching and replication. Contrary to the common delivery approach that dedicates a separate stream (i.e., a resource channel) to each client request, a scalable delivery protocol attempts to serve multiple closely-spaced client requests for the same file by a single server stream.

Two categories of scalable delivery protocols have been proposed. *Periodic broadcast* protocols [3, 115, 65, 59, 45, 87, 35, 88, 89, 57, 56, 76] are based on the following two observations: 1) media file access probabilities are likely to be highly skewed [30]; and 2) clients may be willing to wait tens of seconds for the beginning of playback. The media server simply broadcasts 10 to 20 of the most popular media files on different channels. Clients wishing to receive a media file tune into the channels that are allocated to that file. In order to achieve server bandwidth efficiency, each media file is partitioned into segments, typically of increasing sizes. The first segment is usually very short, at most tens of seconds in length. It is repeatedly broadcast on a dedicated channel such that clients can receive it and begin the playback with only a short delay. Each subsequent segment is fetched while previous segments are being played back, such that it is received in time to be consumed without jitter. Periodic broadcasting protocols differ in how a media file is partitioned into segments, how the segments are allocated to channels, and how clients schedule the receiving of the segments.

Although periodic broadcasting is highly scalable for streaming popular media files, since the server bandwidth usage is independent of the client request rate, it suffers from the following problems:

- it is very inefficient for cold and lukewarm media files;
- the start-up delay imposed by periodic broadcasting protocols may be undesirable in some applications;
- interactive functions, such as fast-forward and rewind, are difficult or impossible to efficiently provide.

The other category of scalable delivery protocols attempts to overcome the above drawbacks by reactively responding to client requests [52, 2, 22, 58, 20, 46, 101, 36, 37, 15, 38, 69, 27]. For each client request, a new stream is created and delivers the requested media file from the beginning, enabling minimal start-up delay. To achieve scalability, these *immediate service* protocols let a client snoop on a previous

stream that is currently transmitting the same media file, and store the data in the client's local buffer. Once the client has received all the data prior to the point in the media file at which it began to snoop on the previous stream, its own stream can be terminated, and the client continues receiving data from the previous stream. Various immediate service protocols have been proposed, including patching [58, 20, 46, 101], tapping [22], adaptive piggybacking [52, 2, 69], hierarchical stream merging (HSM) [36, 15, 38, 27, 14], and bandwidth skimming [37].

Immediate service protocols can easily support interactive functions such as fast-forward and rewind by re-allocating a new stream to the client who initiated the interactive request. Moreover, these protocols do not require prior knowledge of the popularity of the media files being delivered. Compared with periodic broadcast protocols, they work better for streaming lukewarm and cold media files, but are less efficient for hot media files.

Some research has proposed protocols combining elements of these two approaches. The basic idea is to dynamically distinguish between hot and cold media files, and then use periodic broadcast to deliver hot media files and immediate service to deliver cold media files [47]. The difficulties of the hybrid approaches lie in accurately and efficiently identifying hot media files, and in reacting to changes in file popularity.

### 1.1.3 Complex Multimedia

Audio and video normally are compressed before being stored or delivered over the network. Compressed audio files are usually constant bit rate (CBR); i.e., each second of an audio file requires the same number of bits to represent it. Video may use either constant bit rate or variable bit rate (VBR) compression. CBR video files are easily managed and transmitted over the network, but they cannot efficiently provide good quality for motion-intensive scenes. VBR compression allocates bandwidth according to the properties of different scenes; for example, more bandwidth is allocated for motion-intensive scenes. It can maintain quality better across the

whole video, since the peak rate of a compressed VBR file is usually 2 - 3 times higher than the average rate [77], however, such files are often hard to stream efficiently.

*Work-ahead smoothing* has been proposed to smooth a variable bit rate file into a near constant bit rate stream, so as to make delivery across a network easier [99, 78, 41]. Work-ahead smoothing uses available network bandwidth during the delivery of the low bit rate portions of a media file to deliver data in later high bit rate portions. A small start-up delay may be needed to smooth the first several seconds of the stream. Then, work-ahead smoothing can efficiently reduce both the peak rate and the rate variability of the stream without introducing any further latency.

A second type of complex media is *composite* media. As multimedia authoring, editing, and presentation tools become easier to use and more powerful, composite media files that consist of a mix of audio, video, static images, and plain text will become increasingly popular. For example, one can imagine the Olympic games being broadcast live over the Internet, where a typical broadcast might consist of multiple multimedia substreams, including background sounds, video from particular events, close-ups, game statistics, and narrations.

Regardless of the type of the media files being delivered (e.g., audio, video, or composite), the existing scalable delivery protocols all assume *linear* media, consisting of a single sequence of encoded data units (e.g., video frames). All clients accessing the same file will receive all, or a subsequence of, the same sequence.

Although linear media is sufficient for many existing multimedia applications, the widespread adoption of the Internet and the dramatic increase in network capacity may enable new multimedia applications that provide more interactivity and customization through use of more complex media structures. For example, movies are currently limited to a linear structure by the broadcast technology used to deliver them in theatres or on TV. Internet delivery may enable new types of entertainment and educational video, such as multi-ending movies, in which there are multiple threads of story line connected by branch points at which clients dynamically select the path to be followed. Another example is a "virtual tour" where clients choose their own navigation path [124]. One can also imagine that in a future

news-on-demand system, viewers in different cities may be able to watch the same national news followed by different regional news and different local news. Or in a video-on-demand system, different viewers may be given localized and customized advertisements during movie breaks. Although these applications could use a collection of linear media files, for each story line in a multi-ending movie for example, it may be more efficient to recognize the structures inherent in the media threads and exploit them during delivery.

## 1.2 Contributions of the Thesis

This thesis investigates protocol design and performance issues concerning scalable on-demand streaming of stored complex multimedia. The main contributions of this thesis are:

- Tight bounds are developed on the minimum network bandwidth requirement for scalable on-demand streaming. Previous work has considered only server bandwidth requirements. Although for clarity, these bounds are developed in the simple context of constant bit rate linear media files, they are applicable to complex multimedia as well.

- Bounds on the minimum server bandwidth needed for streaming variable bit rate media files are developed, and a scalable immediate service delivery protocol for such media called VBR Bandwidth Skimming (VBRBS) is proposed and its performance is evaluated.

- Server bandwidth bounds are developed for scalable on-demand streaming of non-linear media objects, and practical immediate service and periodic broadcast protocols for such media are proposed and evaluated. As a proof-of-concept, a scalable non-linear media streaming prototype system is developed.

### 1.2.1 Network Bandwidth Bounds

The objective of a scalable delivery protocol is to deliver media files with bandwidth that grows much slower than linearly with the client request rate, or, in the case of periodic broadcast, with the inverse of the client start-up delay. A basic question is how slow this growth can be. Also of interest is the question of how close existing scalable delivery protocols come to achieving the minimum possible bandwidth usage.

Previous research has determined the minimum server bandwidth required by on-demand streaming protocols that guarantee a maximum start-up delay [45, 19, 102] and protocols that provide immediate service [38]. However, the minimum network bandwidth requirement of scalable on-demand streaming has not been studied. The analysis is complicated by the fact that the network bandwidth requirement depends not only on the streaming protocol, but also on the network topology and on the multicast distribution tree that is employed. In this thesis, tight bounds on the minimum required network bandwidth for simple canonical multicast delivery tree topologies are developed. The methods used and the insights gained are then applied to randomly generated large network topologies to analyze and compare the minimum network bandwidth required by on-demand streaming over various multicast distribution trees constructed using existing and new algorithms. The results suggest that simple shortest path trees work reasonably well as multicast distribution trees for scalable on-demand. Their network bandwidth requirement is only modestly more than the network bandwidth required by multicast distribution trees that attempt to optimize network bandwidth usage. It is also demonstrated that it is possible to simultaneously achieve reasonably close to the minimum possible network and server bandwidth usage with a practical immediate service protocol, or with a new variant of periodic broadcast protocol that is proposed in this thesis.

### 1.2.2 Scalable On-demand Streaming of VBR Media

Most of the scalable delivery protocols proposed in prior work assume constant bit rate (CBR) media. However, video may use variable bit rate (VBR) compression in order to efficiently achieve uniform quality. Furthermore, in multimedia streaming applications such as news-on-demand and live broadcast, the various media types (i.e., audio, video, text, and static images) may be combined in a composite multimedia presentation, also resulting in a VBR media file. Because of its intrinsic rate variability, VBR media cannot be managed and delivered as easily as CBR media. Work-ahead smoothing techniques are often used to reduce the peak rate and the rate variability when streaming a VBR file.

In this thesis, a lower bound on the required server bandwidth for on-demand streaming of VBR media is developed, and applied to determine the impact of work-ahead smoothing on scalable streaming. It is found that work-ahead smoothing actually reduces the potential benefits of scalable delivery protocols. This is because those protocols achieve bandwidth reduction largely through sharing the delivery of the later portions of a media file among multiple clients, while work-ahead smoothing moves data from these later portions to earlier portions that are less widely shared, in order to generate a smoother stream.

Based in part on the above analysis, two possible approaches to immediate service delivery of VBR media files are evaluated. One approach is to fully smooth a VBR file into a CBR or near-CBR stream and then apply an existing scalable delivery protocol to the smoothed stream. The other approach is to use a (new) scalable on-demand delivery protocol that exploits the bit rate profile of the VBR file. Such a protocol, called VBR bandwidth skimming (VBRBS), is designed and shown by simulations to outperform, sometimes considerably, the approach wherein an existing immediate service protocol is applied to a fully smoothed VBR file.

### 1.2.3 Scalable On-demand Streaming of Non-linear Media

Current highly-scalable content delivery services such as TV employ a broadcast model where end-users play a passive role in receiving the content. Internet delivery may enable new opportunities for future customized and interactive multimedia applications, such as pick-your-own-ending movie-on-demand, in which the media objects to be delivered include parallel sequences of data units (e.g., video frames), among which clients can dynamically select at designated branch points. Since clients accessing the same non-linear media file may receive different sequences of data units, according to their respective selections at branch points, existing scalable delivery protocols cannot be directly used to deliver such files.

In this thesis, a number of approaches to scalable non-linear media delivery are defined and evaluated. Some of these approaches assume advance knowledge of client path selection at each branch point, either by measurement of the overall client path choice frequencies in the respective system or relying on client classification or pre-selection. Others assume no *a priori* knowledge and must achieve a suitable compromise between aggressive sharing of server transmissions, and client reception of data that turns out to be from a different path than the path the client will select. The minimum required server bandwidth and associated client data overhead, defined as the data that clients receive but never use, of these approaches are analyzed. It is found that fairly accurate *a priori* knowledge of client path selection can enable close to minimal server bandwidth usage as well as substantial client data overhead savings. In the absence of such knowledge, the client data overhead can still be greatly reduced at a relatively small server bandwidth cost, through control of which data clients will potentially receive in advance of knowing whether it is on their path.

Based on the insights gained from the above analyses, new immediate service and periodic broadcast protocols are designed for non-linear media. Variants are developed that assume either having full knowledge of client path selection or having no *a priori* knowledge at all. Simulation results show that the protocols that utilize

advance path knowledge are very efficient in that they can achieve close to minimal server bandwidth usage.

As a proof-of-concept, a simple non-linear media immediate service protocol has been implemented in the SWORD multimedia streaming prototype [109], demonstrating that non-linear media streaming can be realized with relative ease.

## 1.3 Thesis Organization

The remainder of the thesis is organized as follows. Chapter 2 overviews related research on scalable multimedia delivery. The minimum network bandwidth required for on-demand streaming is investigated in Chapter 3. Chapter 4 analyzes the required server bandwidth for variable bit rate media streaming, and proposes and evaluates a new scalable immediate service on-demand streaming protocol for such media. Chapter 5 discusses the minimum required server bandwidth for on-demand streaming of non-linear media, proposes and evaluates practical on-demand delivery protocols for such media, and describes the design and preliminary experimentation with a prototype scalable non-linear media on-demand streaming system. Chapter 6 summarizes the thesis and outlines possible areas for future work.

# Chapter 2

# Related Research

Audio and video are termed *continuous* media because they must be captured or rendered continuously at specified rates. Other distinguishing characteristics compared to plain text and static images include large data volume and high bandwidth requirements. These characteristics and the challenges they pose for scalable streaming are discussed in Section 2.1.

Scalable delivery protocols achieve bandwidth efficiency by sharing one server stream among multiple receivers, which demands one-to-many communication support from the underlying network. In broadcast networks with a shared medium, such as cable networks, satellite networks, and wireless LANs, one-to-many communication can be naturally realized by simply broadcasting to the whole network. In point-to-point networks that do not have a shared medium, such as the Internet, however, multicast (either at the network layer or application layer) has to be employed to deliver a stream from the sender to many receivers. The basic idea is to build a spanning tree covering the sender and all receivers over the underlying network topology. Section 2.2 overviews various methods and algorithms for constructing multicast distribution trees. Section 2.3 summarizes representative scalable delivery protocols that have been proposed, including periodic broadcast protocols, hierarchical stream merging (HSM), and bandwidth skimming (BS). Section 2.4 discusses a tight lower bound on the server bandwidth required for scalable on-demand streaming and the server bandwidth efficiencies of scalable delivery protocols compared to the lower bound.

## 2.1 Continuous Media Overview

Continuous media, i.e., audio and video, forms an increasing fraction of Internet traffic, owing to the widespread use of such network applications as the World Wide Web, peer-to-peer file sharing, voice mail, Internet telephony, video-on-demand, video conferencing, and distance learning. However, video and audio streaming also poses new challenges to the deployment of network services and the development of network applications:

- Video and audio files are typically several orders of magnitude larger than text and image files. For example, a typical 3 minute MP3-encoded song with near-CD quality requires about 3 Mbytes, and a 90 minute MPEG-2 movie occupies about 3 Gbytes. Large files demand systems with large storage capacities, which have become less of a problem in recent years, since multi-terabyte storage systems capable of holding thousands of high-quality (e.g., MPEG-2) videos have become available at a relatively low price.

- Video streaming, in which video data is delivered at a sufficient rate that playback can be concurrent with delivery, requires a particularly high data rate (*bandwidth*). For example, the playback bit rate of an MPEG-1 compressed video can be about 1.5 Mbps (with a resolution of up to $352 \times 240$ pixels/frame with 24 bits/pixel at a frame rate of 30 frames/sec) [110], and a broadcast-quality MPEG-2 encoded video (with a resolution of up to $720 \times 480$ pixels/frame with 8 bits/pixel at 24 frames/sec) normally requires a rate of $4 - 6$ Mbps [79].

- Audio and video are *delay-sensitive*. Once playback begins, each media unit must be rendered at a particular time. If a media unit is not received until after its playback time, it becomes useless. When a continuous media file is streamed over a network, the file data normally experiences significant end-to-end delay and delay variance (i.e., jitter). Techniques for accommodating end-to-end delay and jitter include introducing a start-up delay and using

buffering at the client. For streaming live audio/video without interactivity or streaming stored short video clips, a start-up delay of tens of seconds is usually acceptable. For streaming full-length (e.g., 90 minutes) stored movies, a longer start-up delay of up to several minutes may be tolerable. However, for real-time, interactive applications, such as on-line games or conferencing, a delay longer than hundreds of milliseconds will become frustrating.

- Video and audio are *loss-tolerant* since not all of the file data are required for good quality playback, packet loss during delivery should be minimized, however, since the quality of playback generally degrades with increased loss. In the current best-effort Internet, when congestion occurs, all traffic streams including audio and video may be subject to bursts of packet loss. In streaming applications, timing constraints may not permit retransmissions of lost packets, which could dramatically impair video/audio playback quality.

### Audio

Audio is a common traffic type on today's Internet, owing to applications such as Internet phone [26], instant messaging [126], on-line radio [108], and peer-to-peer file swapping [68].

Audio, especially high-quality music, is typically compressed prior to being stored on a computer system or delivered across the Internet. For example, the raw CD-quality stereo music encoded by the standard PCM (Pulse Code Modulation) technique has a data rate of 1.411 Mbps [71]. MPEG-1 layer 3, more commonly known as MP3 [82], can reduce this rate by a factor of 12, to 128 Kbps for CD-quality stereo music, without losing sound quality. It can achieve an acceptable sound quality at even lower bit rates. For compression of speech, GSM (13 Kbps), G.729 (8 Kbps), and G.723.3 (both 6.4 and 5.3 Kbps) are some popular publicly available standards [71]. There also exist a large number of proprietary audio compression techniques employed by companies such as RealNetworks.

### Video

Video is composed of a sequence of still images (called *frames*) that should be displayed at a constant rate, typically either 25 frames per second (PAL) or 30 frames per second (NTSC). Uncompressed video demands extensive bandwidth. Using compression, bandwidth requirements can be reduced by as much as a factor of 100 with little apparent loss in quality.

Since a video file is made up of still images, it can be compressed simply by compressing each image separately. For example, video capture cards often employ MJPEG (Motion JPEG) [54] for video compression, in which each video frame is independently compressed using JPEG, a compression standard for still images [64]. MJPEG compression only exploits spatial redundancy in a video frame. It does not consider the redundancy between consecutive frames (i.e., temporal redundancy).

The MPEG family of compression standards was developed by the Moving Picture Experts Group (MPEG), a working group of the International Standard Organization and the International Electrotechnical Commission (ISO/IEC). It includes some of the most widely used video compression standards, such as MPEG-1 [105], MPEG-2 [106], and MPEG-4 [83]. MPEG-1 aims at generating VHS (or CD) quality video at a rate of approximately 1.5 Mbps. An MPEG-1 compressed stream can be played back from a single speed CD-ROM with a screen size of $352 \times 240$ pixels (24 bits/pixel) at a frame rate of 30 frames per second [110]. MPEG-2 is a compression scheme for high-quality (e.g., DVD) video and typically yields a data rate of 4 to 6 Mbps. Currently, RealNetworks Real Player [97] and Apple QuickTime [11], two of the most popular multimedia streaming products, can play back MPEG-2 encoded content. MPEG-4 supports layered encoding in which a compressed media file consists of a base layer and one or more enhancement layers that yield a higher quality display. MPEG-4 targets a wide range of data rates. It also features object-oriented compression, rather than frame-based compression as with MPEG-1 and MPEG-2. Object-oriented compression enables individual objects in a compressed media file to be manipulated interactively, for example, using Web-like hyperlinks [51]. The

three most popular multimedia streaming products, RealNetworks Real Player, Microsoft Media Player [80], and Apple QuickTime, all support the MPEG-4 standard, although they also support proprietary video compression techniques.

H.261 and H.263, which are defined by the International Telecommunications Union Telecommunication Standardization Sector (ITU-T), are additional popular video compression standards [54]. H.261 has been defined for video telephony and video conferencing over Integrated Services Digital Networks (ISDN). It is designed for transmission rates that are multiples of 64 Kbps. H.263 has been designed for applications over wireless and Public Switched Telephone Networks (PSTN). It assumes that only very low bit rates are feasible.

### Composite Media

Composite media consists of media objects of various types, such as static images, text files, and video and/or audio clips, organized into meaningful presentation sequences with specified relative viewing/playing time constraints [107, 49]. With the rapid development of multimedia authoring, editing, and presentation techniques and software, composite media may become increasingly popular. For example, in the future, sporting events such as the Olympic games may be broadcast over the Internet, with the broadcast streams consisting of audio and video of one or more concurrent events, close-ups, game statistics, and narrations.

## 2.2 Multicast Service Model

In contrast to the unicast service model, where each communication involves only one sender and one receiver, *multicast* communication is based on a *many-to-many* model where multiple nodes are organized into a group and each communication is to the whole group. Using multicast rather than unicast can more efficiently support applications such as software update distribution, real-time stock quote transmission, video conferencing, scalable streaming, and distance learning.

Compared to unicast, where a linear network path is required between each communicating pair of nodes, multicast requires a distribution tree (a spanning tree) across all members in the multicast group. If a link in the distribution tree leads to multiple group members, only one copy of the data will be transmitted on the link. If, on the other hand, the data was sent using multiple unicast transmissions, multiple copies of the same data will exist on shared links, which clearly is a waste of resources. Perhaps even more importantly, multicast is much more efficient for the sending node.

A number of crucial issues arise in provision and use of multicast, including multicast address allocation, group management, distribution tree generation, security, and pricing [71]. This section will focus on how various kinds of multicast distribution trees are created by multicast routing algorithms.

### 2.2.1 Multicast Distribution Trees

Multicast distribution trees are spanning trees that connect all members in a multicast group. Multicast distribution trees can be classified as either *group-shared* or *source-based*.

With a group-shared tree, all senders to the multicast group use the same multicast distribution tree. The network bandwidth cost can be minimized by using a minimum cost spanning tree, termed as Steiner tree. Although building a Steiner tree is NP-complete [48], very efficient approximation algorithms have been proposed [117, 70, 120, 121]. However, none of the existing multicast routing algorithms use Steiner trees because of the high overheads to build and maintain them as nodes join and leave the multicast group, and since simpler approaches give similar performance [122].

A type of group-shared tree, which is used by several multicast routing protocols, is the centre-based tree [12]. Centre-based trees are constructed by first identifying a root or "centre" node (also known as a rendezvous point or a core). Then, a multicast distribution tree is built through a process wherein each of the multicast receivers

adds the links on its unicast path towards the centre (determined by the unicast routing algorithm) to the multicast tree. Although centre-based trees are much simpler than Steiner trees, they have the disadvantage of requiring identification of an appropriate centre node.

When source-based trees are used, a separate tree is created for each sender to the multicast group. Dijkstra's algorithm [112], which builds a spanning tree with minimum cost from a source node to all destinations, can be used to generate source-based trees. This is the approach used in MOSPF, a multicast routing algorithm used in Internet autonomous systems (ASs) that use the OSPF unicast routing algorithm [81]. A second approach to building source-based trees is reverse path forwarding (RPF), and is used in the DVMRP multicast routing protocol [116].

The basic idea of RPF is as follows: each node forwards a multicast packet on all of its outgoing links (except the one on which the packet was received) if and only if the link on which the packet was received is on the shortest path from itself to the sender. All multicast packets received from other links are discarded. This algorithm creates a distribution tree that is composed of all shortest paths from all receivers back to the sender. When network paths are symmetric, these reverse shortest paths are also the shortest paths from the sender to all receivers. One potential problem with RPF is that a node may forward packets to all its downstream nodes even though no downstream nodes have joined the multicast group. This problem can be solved by pruning, in which a node that does not have downstream receivers sends a prune message to its upstream node, causing that node to no longer forward it the packets being sent to the respective multicast group. A node that receives prune messages from all of its downstream nodes will in turn send a prune message to its upstream node.

A third approach to building source-based trees is based on each receiver sending a message to each source which causes the links on the unicast path back to the respective source to be added to the respective tree, and is used in the PIM-SM multicast routing protocol [32, 39]. Protocol independent multicast (PIM) identifies two multicast distribution scenarios, namely, *dense mode* and *sparse mode*.

PIM dense mode (PIM-DM) assumes that a large majority of network nodes are participating in the multicast session. Therefore, it uses a variant of the reverse path forwarding approach. PIM sparse mode (PIM-SM) assumes that only a small number of network nodes are participating in the multicast session. Rather than flooding messages all over the network, PIM-SM initially creates a centre-based tree, and then as each source becomes active, receivers send these source join messages that create source-specific trees. PIM-SM is used together with the MBGP [16] and MSDP [44] protocols in the current Internet architecture for inter-domain multicast routing [8].

### 2.2.2 Source-specific Multicast

The multicast service model currently supported at the network layer in the Internet (i.e., IP multicast) is an "any-source" model in which any node (not necessarily been a member of the group) can send to a multicast group [31]. New multimedia applications such as audio/video streaming of stored content usually involve only one sender (i.e., the server) sending media files to a group of receivers (i.e., the clients). Therefore, these application can be well supported by a much simplified multicast model.

Derived from EXPRESS [55], IETF has developed a new multicast service model called *source-specific multicast* (SSM), which supports only one-to-many delivery applications [18]. Both IPv4 and IPv6 address ranges have been allocated specifically for SSM [53, 60]. Since in SSM, the source is the only sender, the routing algorithms that build the multicast distribution tree across the sender and all receivers can be much simplified. For example, in PIM-SM, which is the most commonly used group-shared tree routing algorithm, the initial use of a rendezvous point and a group-shared tree is no longer necessary. Instead, receivers can immediately send join messages to the (single) source so as to build a (reverse) shortest path tree rooted at the source [43].

### 2.2.3 Application Layer Multicast

A number of crucial technical problems, such as scalability, reliability, security, heterogeneity, inter-domain multicast routing, and address allocation, remain all or partially unresolved for IP multicast. Because of the difficulties and complexities in solving these problems at the network layer, *application layer multicast* has been proposed as an alternative approach to providing a one-to-many (or many-to-many) communication service [23, 24, 62, 90, 129]. Application layer multicast leaves the network layer service as simple as possible, and shifts the responsibility for supporting multicast service to network infrastructure servers operating at the application level (e.g., CDN servers), and/or the end hosts themselves.

The basic idea of application layer multicast is to build a virtual network, called an *overlay*, over all participating nodes. Each virtual link from one node to another in the overlay network is actually a unicast path in the underlying network. Use of multicast requires that a multicast distribution tree be constructed from the sender to all member nodes on top of the overlay. Multicast packets are then transmitted along the distribution tree, replicated and forwarded by the nodes at every branch point, until they reach every receiver.

Use of application layer multicast relieves the underlying network from the task of supporting one-to-many (or many-to-many) communication. However, it has its own problems, such as that of constructing an efficient overlay network across all participating nodes, and of effectively maintaining the overlay network when network conditions change and when nodes join or leave the overlay.

## 2.3 Scalable Delivery Protocols

A number of scalable delivery protocols have been proposed that substantially reduce the server and network bandwidth required for streaming popular multimedia files to a large number of concurrent clients. The basic idea is to merge clients making closely-spaced requests for the same file into groups and broadcast/multicast a server stream to each group. This idea has its roots in previous work on scalable download

of a number of small objects to multiple clients using broadcast/multicast [34, 10, 123]. The main problem considered there is to determine a transmission schedule of objects such that the average client delay to completely receive a required object is minimized, whereas the primary concern of scalable streaming media delivery protocols is to minimize the server bandwidth usage and client start-up delay while ensuring continuous playback.

Scalable streaming media delivery protocols can be broadly classified as *server push* or *client pull*. The server push protocols, also known as *periodic broadcast* protocols [3, 115, 65, 59, 45, 87, 35, 88, 89, 57, 56, 76], allocate a fixed number of server channels per media file and periodically broadcast the file data on the dedicated channels. Clients wishing to receive a file tune in to the corresponding channels for that file. Since these protocols schedule transmissions of file data without consideration of what client requests are currently being served, they can be an efficient technique for delivering the most popular media files. However, these protocols normally introduce a start-up delay before a client can begin playback of a media file. Moreover, it is hard for these protocols to provide seamless and continuous interactive functions, such as fast-forward and rewind.

Client pull protocols initiate server streams only in response to client requests. *Batching* is the simplest form of such protocols, in which client requests for the same file that are received by the server within a short time interval are queued, and then are served by a single multicast stream [29, 1, 113]. Although efficient queuing policies are employed, batching yields relatively high client start-up delay.

*Immediate service* protocols [52, 2, 22, 58, 20, 46, 101, 36, 37, 15, 38, 69, 27] have been proposed that schedule a separate server stream upon the arrival of each client request so as to allow clients to begin playback without any substantial start-up delay. To make the streaming scalable, clients whose requests for the same file are made close together in time are merged into one group and share a single server stream. Although immediate service protocols use server and network bandwidth that increases with request rate, unlike periodic broadcast protocols, they are efficient for low to moderate request rates and can easily support interactive functions.

Section 2.3.1 overviews representative periodic broadcast protocols. Sections 2.3.2 and 2.3.3 introduce two representative immediate service protocols, namely, hierarchical stream merging (HSM) and bandwidth skimming, respectively.

### 2.3.1 Periodic Broadcast Protocols

A streaming server employing a periodic broadcast protocol schedules transmissions of file data without consideration of what client requests are currently receiving service. Each media file is partitioned into a sequence of segments, typically with increasing sizes. Each segment is periodically multicast, usually on its own channel, with the next transmission beginning immediately after the end of the previous one. A client may have to wait for the beginning of the next broadcast of the first segment before starting to receive and play back the file data, or in other protocols of this type, may begin to receive immediately, and start playback once reception of the first segment is complete. The start-up latency before playback can begin can vary from seconds to minutes for different protocols, and depends on the number and rates of the channels allocated to the file. Reception of subsequent segments occurs in parallel with reception of the first segment and/or playback, such that each segment's data is received prior to its play point.

The major advantage of periodic broadcast protocols is that they can accommodate high request rates extremely well, since the server bandwidth required for delivery of a media file is fixed regardless of the rate of client requests for the file. This property makes periodic broadcast protocols especially appropriate for distributing heavily demanded hot media files. On the other hand, periodic broadcast protocols are very inefficient for delivering cold media files since frequently resources are wasted with segment transmissions for which there are no receivers. Moreover, the proactive nature of periodic broadcast protocols makes it very hard for them to support interactive functions that entail jumping forward in the media file.

Examples of periodic broadcast protocols include pyramid broadcast [115], permutation-based pyramid broadcast [3], skyscraper broadcast [59, 45, 35], harmonic broadcast [65] and its variants [87, 88], greedy equal bandwidth broadcast [57, 56], and optimized periodic broadcast [76]. These protocols differ in the partitioning of a media file, the assignment of segments to channels and the transmission rates on these channels, and the segment reception schedules of clients.

The optimized periodic broadcast (OPB) protocol [76] assumes that clients are able to simultaneously listen to a maximum of $s$ channels, each delivering data from a different file segment at a rate $r$. Each media file is partitioned into $M$ segments such that if a client begins listening to the $s$ channels delivering the first $s$ segments immediately upon making its request for the file, begins listening to the channel transmitting segment $m$ ($m > s$) immediately after segment $m - s$ is fully received, and begins playback after the first segment is received in its entirety, then every segment can be received in its entirety just before the required playback time of the beginning of the segment. The (deterministic) start-up delay that clients experience is the time required to download the first segment.

Figure 2.1 illustrates the operation of the OPB protocol assuming that the media file is partitioned into six segments (i.e., $M = 6$), the channel streaming rate is equal to the file playback rate (i.e., $r = 1$), and a client is able to receive transmissions on two channels simultaneously (i.e., $s = 2$). The shaded area for each channel indicates the period of time during which an example client, which arrives at the point indicated in the figure, listens to that channel. The client starts the reception of the first two segments immediately. The client starts reception of segment $m$ ($m > 2$) immediately after completing reception of segment $m - 2$. As illustrated in the figure, the client starts to play back each segment right after it has been fully downloaded.

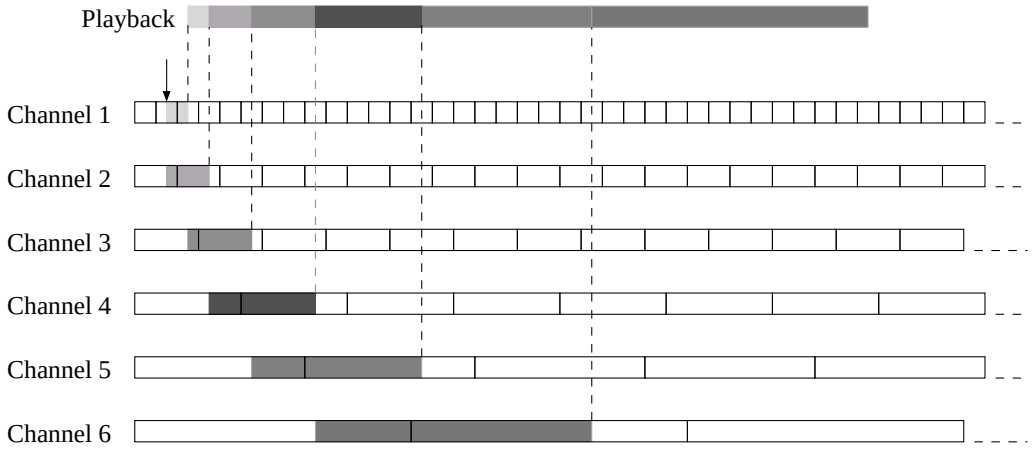



Figure 2.1: Optimized Periodic Broadcast Protocol
($M = 6$, $r = 1$, $s = 2$)

The size of the $m^{th}$ segment ($m \geq 2$) can be computed using

$$\frac{l_m}{r} = \begin{cases} \frac{l_1}{r} + \sum_{i=1}^{m-1} l_i & 1 < m \leq s \\ \sum_{i=m-s}^{m-1} l_i & m > s, \end{cases} \tag{2.1}$$

where $r$ is the segment streaming rate [76]. This expression reflects the fact that for any of the first $s$ segments, the maximum time that can be allowed to receive the segment $m$ is equal to the time required to receive the first segment plus the sum of the playback times of the first $m-1$ segments. For any subsequent segment $m$, the maximum time from the beginning of reception of that segment until reception is complete is equal to the sum of the playback times of segments $m-s$ to $m-1$. Using the above equation, the sizes of the $2^{nd}$ through $5^{th}$ segments in Figure 2.1, relative to the size of the first, are 2, 3, 5, 8, and 13, respectively.

The server bandwidth required to deliver a media file partitioned into $M$ segments, each transmitted at rate $r$, is $r \times M$. The deterministic start-up delay (i.e., the time required to download the first segment) is $l_1/r$. The computed segment size progression is such that the required value of $M$, and thus the server bandwidth, grows only *logarithmically* with the inverse of the desired start-up delay. This is

in contrast to simpler techniques such as staggered broadcast [9] that do not segment the media file, which require server bandwidth that increases linearly with the inverse of start-up delay.

### 2.3.2 Hierarchical Stream Merging

Hierarchical stream merging (HSM) [36, 38] is a family of immediate service protocols with required server bandwidth that grows only *logarithmically* with file request rate, unlike conventional unicast delivery that can also provide immediate service, but for which the required server bandwidth is linear in the request rate. The simplest variant of HSM requires that each client be capable of receiving two server streams, each at the file playback rate, simultaneously. Upon receiving a client request for a file, the streaming server immediately initiates a new stream delivering data from the beginning of the file, so that the client can start playback essentially immediately (some small start-up delay may be required to accommodate possible network delay jitter). At the same time, the server instructs the client to snoop on a previous stream (i.e., the merge target stream) which is delivering the same file to a previous client or a group of clients, if such a stream exists. Note that streams with multiple receivers would need to be delivered using some multicast mechanism. Once the later client has received all of the file data prior to that obtained by snooping, its own stream can be terminated, and the client is said to be "merged" with the client or group of clients listening to the merge target stream. This merged group of clients can then go on to merge with other groups of clients, yielding a hierarchical merging structure. In this manner, the server bandwidth and network bandwidth required to deliver the file are reduced.

Various hierarchical stream merging protocols differ in whether the merge target stream is determined statically [15, 27] or dynamically [36]. When a merge target stream is chosen on-the-fly, a number of different strategies can be employed. For example, with *earliest reachable merge target* (ERMT), the closest stream with whose clients a group of clients $G$ could merge is chosen as the merge target stream.

Whether or not this particular merge actually occurs, however, depends on whether or not some later group of clients merges with $G$ before $G$ is able to merge with its target group. ERMT takes into account such race conditions, but can consider only the groups of clients present at each particular point in time. Subsequent client request arrivals may change the merging structure. A somewhat simpler strategy that does not require consideration of race conditions is *closest target* (CT). With CT, at each point in time, each client simply snoops on the closest earlier stream delivering the same file, even though this stream might be terminated prior to the merge (thus making the merge impossible) owing to the target group itself accomplishing a merge with some earlier group.

Figure 2.2 illustrates an example of HSM (applicable to both ERMT and CT). On the x-axis is time, and on the y-axis file position. Each of the clients $A$, $B$, $C$, and $D$ receives a new stream from the server so as to allow immediate service, when making those requests for the file at times $T0$, $T1$, $T3$, and $T4$, respectively. Each of the clients $B$, $C$, and $D$ also listens to an earlier stream. Clients $B$ and $C$ snoop on client $A$'s stream, and client $D$ listens to client $C$'s stream. The dashed lines in the figure illustrate the total amount of data accumulated by each client. At time $T2$, client $B$ has received all of the file data that it missed from client $A$'s stream, and is merged with client $A$. Client $B$'s stream is terminated, and clients $A$ and $B$ continue listening to client $A$'s stream (labelled now stream $AB$ in the figure). At time $T5$, client $D$ merges with client $C$. Clients $C$ and $D$ continue to snoop on stream $AB$ until time $T6$ when the latest client in the group (client $D$) has received all of the file data prior to the point at which it began listening to stream $AB$, at which point the four clients are merged into one group.

### 2.3.3 Bandwidth Skimming

Bandwidth skimming [37] is a variant of hierarchical stream merging that permits the playback bit rate of a media file to be a greater fraction of a client's maximum sustainable reception rate (or client receive bandwidth *crb*) than the variant

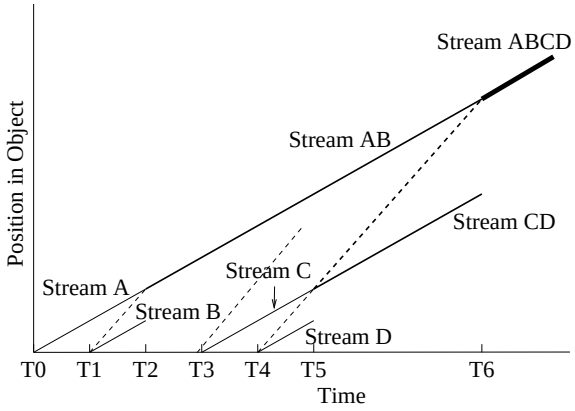



Figure 2.2: Hierarchical Stream Merging

of HSM described above, in which this fraction can be at most 50%. Several variants of bandwidth skimming have been proposed. Among them, *latest patch* has the advantage of simplicity, and also often has comparable performance to the most efficient variant, termed *partition*.

### Latest Patch

In one implementation approach for latest patch, the rate of each server stream (termed here a *delivered stream*) is equal to the client receive bandwidth (*crb*), which can be as little as only a small fraction greater than the file playback bit rate. The stream of data consumed during playback is termed here the *raw stream*. *Latest patch* makes the conservative assumption that the rate of the raw stream is the maximum possible, specifically the file playback bit rate. Client aggregation in latest patch occurs when a later client's delivered stream catches up to a previous client's raw stream, in which case the delivered stream of the previous client can be terminated, and that client can switch to listening to the delivered stream of the later client.

Figure 2.3 presents an example of *latest patch*. Each of the clients $A$ and $B$ receives a stream from the server upon making a request for the (raw) file, at the crb (assumed the same for both clients). The dashed lines illustrate the assumed progress of the raw stream (i.e., the playback) of each client. At time $t$, client $B$'s

delivered stream reaches the same position $P$ in the file as client $A$'s raw stream. Clients $A$ and $B$ are merged into one group and continue listening to the stream previously delivered just to client $B$. Client $A$'s delivered stream is terminated. The raw stream of a group is defined as the raw stream of the earliest client in that group, since that client may be furthest ahead in its playback. In the scenario of Figure 2.3, the raw stream of the group consisting of clients $A$ and $B$ is the raw stream of client $A$. Note that the data that was received by client $A$ beyond the merge point, by client $A$'s delivered stream, is *wasted* in the sense that it will be delivered again by client $B$'s stream, which client $A$ can switch to listening to following the merge point.

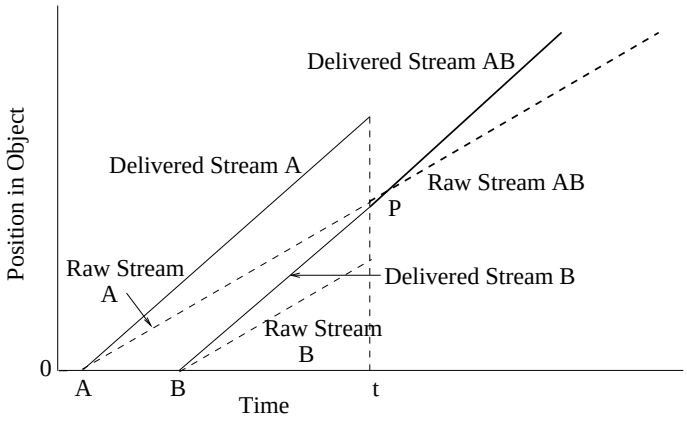



Figure 2.3: Latest Patch

### Partition

The required client receive bandwidth that the *partition* approach uses is the largest value of the form of $(1 + 1/k)$ times the file playback rate, for positive integer $k$, that is no larger than the actual client receive bandwidth. Upon receiving a client request for a file, the server delivers the requested data to the client via $k$ substreams (defined through association with different multicast group, for example), each at a rate equal to $1/k$ of the file playback rate. A later client simultaneously listens to $k + 1$ substreams and goes through $k$ phases to catch up to the previous client. In the first phase, this client listens to all $k$ substreams of its own stream and plays

back the data received. It also listens to one of the previous client's $k$ substreams and stores the data in its local buffer. During phase two, this client listens to only $k-1$ substreams of its own stream because it has already received and buffered the data transmitted in one substream, during the previous phase. Therefore, it can listen to two substreams of the previous client's stream during this second phase. This procedure continues until phase $k$ where the client listens to only one of the substreams of its own stream and snoops on all of the substream of the previous client. At the end of this phase, the clients can be merged and the later client's stream terminated completely. Note that the server needs not transmit substreams that no client is listening to, so the number of transmitted substreams of the later client's stream actually decreases by one after each phase.

Figure 2.4 illustrates an example of the *partition* protocol for $k = 3$, in which case each client is capable of receiving four substreams simultaneously, each at a rate equal to 1/3 of the file playback rate. Initially, the server sends the data via three substreams to client $A$. A later client $B$ requires three phases to catch up to client $A$. In the first phase, client $B$ listens to all three substreams of its own stream and one of client $A$'s substreams. The dashed line indicates the total amount of data accumulated by client $B$. In phase two, client $B$ listens to two of its substreams and two of client $A$'s because it has received the data to be delivered by its third substream. During the last phase, client $B$ listens to only one of its substream and all three of client $A$'s substreams. At the end of this phase, clients $A$ and $B$ can be merged. After the aggregation, client $B$'s stream is terminated, and the group (including clients $A$ and $B$) shares the stream originally delivered to client $A$.

Note that a later client could use either ERMT or CT to determine the target stream to merge with. In this thesis, if not specified, ERMT is employed. Results with CT (not presented) exhibit the same trends.

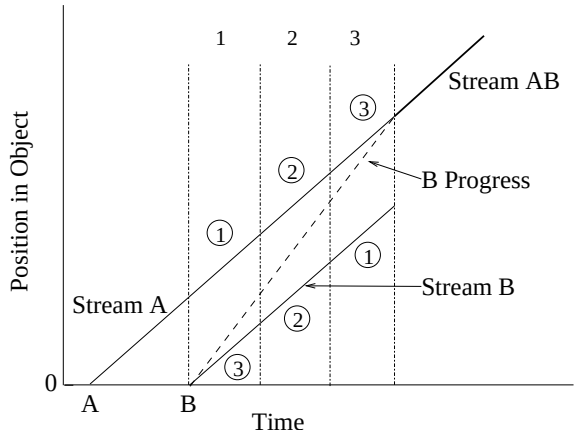



Figure 2.4: Partition
($K = 3$)

## 2.4 Bandwidth Scalability Bounds

With unicast delivery, the server and network bandwidth requirements for on-demand streaming are *linear* in file request rate. With use of multicast, the minimum required server bandwidth for on-demand streaming of a single media file, using the notation defined in Table 2.1, is given by [19, 38, 45, 102]:

$$B_{min}^{srv} = \int_0^T \frac{1}{x + d + 1/\lambda} dx = \ln\left(\frac{T}{d + 1/\lambda} + 1\right) = \ln\left(\frac{N}{ND + 1} + 1\right). \qquad (2.2)$$

Here it is assumed that each client request is for a full-file playback, client request arrivals are Poisson, clients can receive an arbitrary number of arbitrarily short streams simultaneously, and clients have sufficient buffer space to store up to the entire file ahead of playback. The bound is derived by considering a small portion of the file at some arbitrary time offset $x$ from the beginning of the file. Suppose that this portion of the file is multicast at some time $t$. At best, all clients that have previously requested the file and have not yet received this portion will receive this multicast. Assuming Poisson arrivals, the average time until the next request for the file is $1/\lambda$. Another multicast must be scheduled no later than $d + x$ after the next request, so that the respective client will receive the data in time to ensure

Table 2.1: Notation for Server Bandwidth Bounds

| Symbol | Definition |
|---|---|
| $\lambda$ | Average total request rate for a file |
| $T$ | File playback duration |
| $N$ | Normalized file request rate; average number of requests arriving during period of length $T$ ($N = \lambda T$) |
| $d$ | Maximum client start-up delay |
| $D$ | Client start-up delay expressed as a fraction of the file playback duration $T$ ($D = d/T$) |
| $b$ | Maximum sustainable client receive bandwidth (in units of the file playback bit rate) |
| $B_{min}^{srv}$ | Minimum required server bandwidth (in units of the file playback bit rate) |
| $B_{min}^{d=0}$ | Minimum required server bandwidth for any on-demand streaming protocol providing immediate service |
| $B_{min}^{d=0,crb=2}$ | Minimum required server bandwidth for any on-demand streaming protocol providing immediate service and client receive bandwidth of twice the file playback bit rate |
| $B_{min}^{d>0}$ | Minimum required server bandwidth for any on-demand streaming protocol providing bounded-delay service |

jitter-free playback. Therefore, the minimum frequency at which the data at time offset $x$ must be multicast is $\dfrac{1}{x + d + 1/\lambda}$, which yields the bound in equation (2.2).

Although the bound can be generalized to cover a wide class of non-Poisson arrival processes, such as heavy-tailed interarrival time distributions, with difference bounded by a constant independent of $\lambda$ [38], this thesis focuses on Poisson arrivals, as have been observed in a media server measurement study [7].

The minimum server bandwidth required by any on-demand streaming protocol providing immediate service, as analyzed in [38], can be derived from equation (2.2) by setting the start-up delay $d$ to 0, yielding:

$$B_{min}^{d=0} = \ln\left(\frac{N}{ND+1} + 1\right)\Big|_{D=0} = \ln(N+1)\,. \tag{2.3}$$

This minimum server bandwidth grows *logarithmically* with the file request rate.

A conjectured asymptotic (i.e., for high request rate) bound on the required server bandwidth has also been derived for the case in which the client receive bandwidth is bounded [38]. Specifically, for client receive bandwidth equal to twice the file playback bit rate, the conjectured asymptotic bound is given by:

$$B_{min}^{d=0,crb=2} \approx 1.62 \ln(\frac{N}{1.62} + 1), \tag{2.4}$$

where 1.62 is an approximation of $(1 + \sqrt{5})/2$.

Figure 2.5 compares the server bandwidth requirements of HSM-ERMT and HSM-CT with the conjectured asymptotic bound for a client receive bandwidth of twice the file playback bit rate, as given in expression (2.4), and the lower bound assuming unlimited client receive bandwidth, as given in equation (2.3). The y-axis is the server bandwidth expressed in units of the file playback bit rate. The x-axis is the normalized file request rate $N$, equal to the average number of requests arriving during a period of time equal in length to the file playback duration. The figure shows that HSM can achieve close to the minimum server bandwidth usage. Moreover, even though CT is somewhat simpler than ERMT, its associated bandwidth requirement is only slightly worse than that of ERMT.

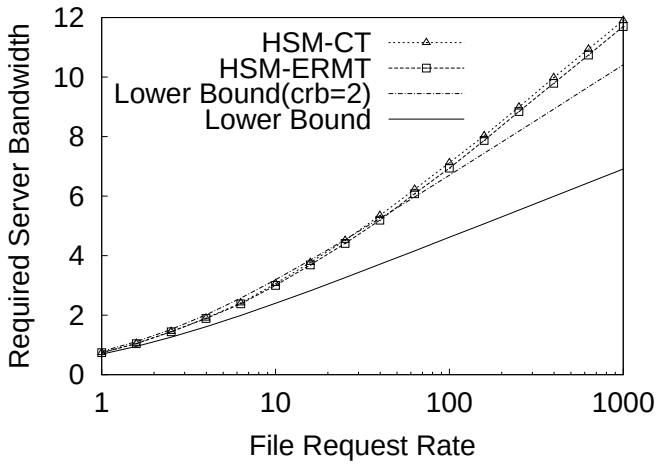



Figure 2.5: Server Bandwidth Efficiency of Hierarchical Stream Merging

The minimum server bandwidth required by any on-demand streaming protocol providing bounded start-up delay service for arbitrarily high client request rate, as developed in [19, 45, 102], can be derived from equation (2.2) by assuming $\lambda \to \infty$, yielding:

$$B_{min}^{d>0} = \ln\left(\frac{T}{d+1/\lambda}+1\right)\Big|_{\lambda\to\infty} = \ln\left(\frac{T}{d}+1\right) = \ln(\frac{1}{D}+1). \tag{2.5}$$

Note that this minimum server bandwidth grows only *logarithmically* with the inverse of the client start-up delay.

# Chapter 3

# Network Bandwidth for Scalable On-demand Streaming

Scalable delivery protocols for on-demand streaming achieve bandwidth reductions by dynamically aggregating clients that make closely-spaced requests for the same file into groups and *broadcasting* or *multicasting* a server stream to each group. When using scalable delivery protocols to stream multimedia files over *true broadcast* networks, such as cable or satellite networks, the same reductions can be achieved for network bandwidth. However, when delivering multimedia files over the Internet or any fundamentally point-to-point network, where *multicast* (either application or network-level) has to be used for one-to-many transmission, the achieved reductions in network bandwidth may differ from those for server bandwidth.

This chapter investigates the network bandwidth savings for scalable on-demand streaming using multicast delivery. The remainder of this chapter is organized as follows. Section 3.1 presents background on the network bandwidth cost of multicast delivery and on-demand streaming. Section 3.2 derives tight bounds on the minimum required network bandwidth for on-demand streaming on simple canonical multicast delivery tree topologies. Section 3.3 applies the analysis techniques developed in the previous section to study the potential network bandwidth savings on multicast delivery trees constructed by alternative algorithms, in large randomly generated networks with random client locations. Section 3.4 presents the network bandwidth requirements of practical immediate service protocols, and discusses whether it is possible for these protocols to achieve close to the minimum network and server bandwidth usage simultaneously. Section 3.5 proposes a new variant of periodic broadcast protocol, OPB/tou, and compares the server and network bandwidth re-

quirements of this protocol against the lower bounds and those of immediate service protocols. Section 3.6 summarizes the work presented in this chapter.

## 3.1 Multicast Delivery Network Bandwidth

The total network bandwidth cost for streaming a file over a fixed multicast delivery tree (i.e., to a fixed set of clients) is the sum of the network bandwidth cost on each tree link, which can be computed by taking the product of the streaming rate, the streaming duration, and the unit network bandwidth cost on the link. Prior work has compared the total network bandwidth cost of various multicast delivery trees constructed over randomly generated network topologies, with each link being assigned a random number as its cost [122]. It was found that approximate minimum spanning trees (Steiner trees) can reduce network bandwidth cost only moderately from that of shortest path trees, with reductions of at most 40% in the cases examined. Other work has studied scaling properties of multicast trees assuming unit cost for each link [91, 25]. Under this assumption, the total network bandwidth cost for transmitting a multicast stream is proportional to the number of links in the delivery tree. Results suggest that if the multicast delivery tree is a shortest path tree, the number of links scales approximately as $m^{0.8}$ where $m$ denotes the number of receivers.

There has been little prior work on the network bandwidth cost of scalable ondemand streaming. One previous study has investigated the network bandwidth usage using unicast delivery [85]. The delivery network is modelled as a $k$-ary tree, with the server at the root, the routers at the intermediate nodes, and the receivers (i.e., clients) at the leaves. For each client request for a file, the server dedicates a stream to transmitting the file data from the beginning to the end, along the path from the root to the corresponding leaf node. The network bandwidth cost on a tree link for delivering a file is assumed to be proportional to the number of streams owing over that link.

Determining the network bandwidth cost for scalable on-demand streaming using multicast delivery is much more complex. In particular, in immediate service protocols, the number of multicast streams varies over time. Furthermore, in both immediate service and periodic broadcast protocols, the clients that listen to the same multicast stream may also vary over time owing to the aggregation of clients or client groups into larger multicast groups, as dictated by delivery protocols. If the multicast delivery tree associated with each server stream is established independently, it may be difficult to construct it efficiently because the set of clients served varies over time. In this thesis, an alternative approach is employed, where a single multicast delivery tree is created and shared by all server streams, with each stream using a subset of the tree links depending on which clients are currently listening to the stream. This approach might be used with an application-level multicast infrastructure or overlay network, as discussed in [23, 24]. In such a network, each application-level server corresponds to a client in the multicast delivery tree and serves its own community of end users. Note that although the analyses and the experiments presented in this chapter assume a shared delivery tree, the minimum network bandwidth results in Section 3.2 and the network bandwidth usage for shortest path trees in Section 3.3 can provide insights for the case of independently constructed delivery trees.

In the context of on-demand streaming over a single shared multicast delivery tree, the network bandwidth cost is no longer directly determined by the number of tree links, because each stream may traverse only a subset of these links. Instead, the network bandwidth cost is a function of both multicast delivery tree topology and client request rate. Steiner minimal trees are not necessarily optimal. For example, when delivering unpopular files (with very low client request rate), shortest path trees become optimal (assuming that path "length" corresponds to a unit of endto-end bandwidth), since there is very little sharing of streams among clients, and therefore the total network bandwidth usage is minimized if the cost of delivery to each client is minimized.

## 3.2 Analysis for Minimum Network Bandwidth

This section considers the minimum network bandwidth requirements on three simple canonical multicast delivery tree topologies. Section 3.2.1 defines the performance metric and the analysis assumptions. Section 3.2.2 and Section 3.2.3 analyze two multicast tree topologies that capture the extreme points with respect to the scaling of the network bandwidth requirement with the number of client sites. Section 3.2.4 considers an intermediate topology that might represent the type of scaling behaviour expected in practice. These simple tree topologies can be combined to form arbitrary tree topologies, and so the analysis in this section can be applied to general delivery trees.

### 3.2.1 Performance Metric and Analysis Assumptions

Of interest in this study is the network bandwidth cost for scalable on-demand streaming. How network bandwidth is priced is an orthogonal issue that is not discussed in this chapter. This study assumes simply that the cost of a unit of network bandwidth can be determined for each multicast delivery tree link that a stream traverses. A link may represent a physical link, an application-level overlay link that maps to a network path, or a route connecting two autonomous systems. For ease of obtaining insight, the derivations and experiments in this study assume further that the cost of a unit of network bandwidth is identical for all links. The analyses can be generalized easily to cases with varying unit network bandwidth costs.

The required network bandwidth is measured in units of the file playback bit rate times the average end-to-end unit bandwidth cost. The latter factor is motivated by the wish to study the scaling of the network bandwidth requirement with the number of client sites, and to compare the network bandwidth efficiencies of various multicast delivery tree topologies. Since the average end-to-end unit bandwidth cost is a function of the average network distance from the server to a client, and the average network distance varies with the network topology and the number of

client sites in the artificial networks considered in the analysis, the average network distance factor (thus the average end-to-end unit bandwidth cost) should be removed from the network bandwidth usage so as to make fair comparisons.

This study considers the network bandwidth requirement for delivery of a single file, assuming that each request is for playback of the file from its beginning to its end. Table 3.1 summarizes the notation used in the following analyses. It is assumed that the client arrival process is Poisson.

Table 3.1: Notation for Minimum Network Bandwidth Analysis

| Symbol | Definition |
|---|---|
| $\lambda$ | Average total request rate for a file |
| $T$ | File playback duration |
| $N$ | Normalized file request rate; average number of requests during period of length $T (N = \lambda T)$ |
| $\lambda_k$ | Average request rate at client site $k$ |
| $N_k$ | Average number of requests from site $k$ during period of length $T (N_k = \lambda_k T)$ |
| $K$ | Number of client sites |
| $d$ | Maximum client start-up delay |
| $D$ | Client delay expressed as a fraction of the file playback duration $(D = d/T)$ |
| $B_{min}^{net}$ | Minimum required network bandwidth cost (in units of the file playback bit rate times the average end-to-end unit bandwidth cost) |

### 3.2.2 Shared Link with Fan-out $K$ Topology

Figure 3.1 shows a delivery tree topology that consists of a shared link and K links that each serves a distinct client site. Each client site corresponds to a particular organizational or geographic group of clients, as served by an application-level multicast server for example. Assuming Poisson request arrivals at each client site, the client request arrivals on the shared link also constitute a Poisson process, with the request rate equal to the sum of the request rates at the client sites, that is, $\lambda = \sum_{k=1}^{K} \lambda_k$, or $N = \sum_{k=1}^{K} N_k$.

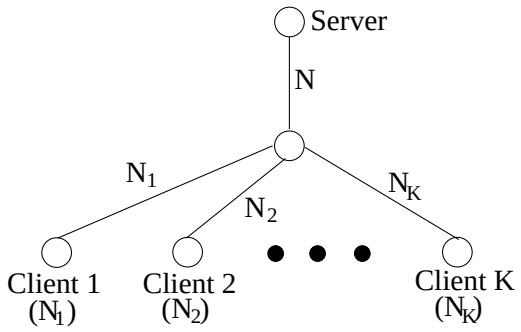



Figure 3.1: Shared Link with Fan-out $K$ Topology

It has not been possible to determine the exact minimum (i.e., optimal) network bandwidth requirement for on-demand streaming for this or the other topologies that are considered here. Instead, lower and upper bounds on the minimum required network bandwidth are derived. The lower bound for each of the topologies considered here can be derived by treating each tree link in isolation and summing the minimum required network bandwidth on each link. The minimum required network bandwidth on a link with request rate $\sum_{i\in\mathcal{C}} \lambda_i$, where $\mathcal{C}$ denotes the set of indices of the client sites served by the link, can be derived by following the same approach used for analyzing the minimal server bandwidth, as discussed in Section 2.4, yielding:

$$\int_0^T \frac{1}{x + d + 1/\sum_{i\in\mathcal{C}} \lambda_i} dx = \ln\left(\frac{\sum_{i\in\mathcal{C}} N_i}{D\sum_{i\in\mathcal{C}} N_i + 1} + 1\right).$$

The lower bound on the minimum required network bandwidth for the shared link with fan-out $K$ topology is thus:

$$B_{min}^{net} > \frac{1}{2}\left(\ln\left(\frac{N}{ND+1}+1\right) + \sum_{k=1}^{K} \ln\left(\frac{N_k}{N_k D+1}+1\right)\right), \tag{3.1}$$

noting that the average end-to-end unit bandwidth cost is equal to $\frac{\sum_{k=1}^{K} 2}{K} = 2$ times the unit bandwidth cost on each link.

Assuming fixed request rate at each client site (i.e., fixed $N_k$ and bounded relative differences among the $N_k$), and fixed maximum client delay $D$, the above lower bound is $\Theta(K)$. For fixed number of client sites $K$ and varying the file request rate $N$ or client delay $D$, the lower bound in relation (3.1) is $\Theta\left(\ln \frac{N}{ND+1}\right)$.

The lower bound in relation (3.1) is not feasible because minimizing the required network bandwidth on the shared link requires each multicast transmission to be shared by all active clients from all sites, while minimizing the required network bandwidth on each dedicated link requires each multicast transmission to be carried on a dedicated link only if that transmission is "just in time" for some client at the associated site (implying, in the absence of simultaneous requests, that each multicast is shared by only the clients from a single site). These two goals cannot be achieved simultaneously.

An *upper bound* on the minimum required network bandwidth for on-demand streaming for the topology of Figure 3.1 can be derived through the analysis of the policy that minimizes the network bandwidth usage on the shared link, at the cost of greater than minimal network bandwidth usage on each of the dedicated links. (The policy that minimizes the network bandwidth usage on each of the dedicated links, at the cost of increased required bandwidth on the shared link, also yields an upper bound, but one that is not as tight.) In the policy that minimizes bandwidth usage on the shared link, a multicast of an infinitesimally small data portion $\Delta x$ at position $x$ is transmitted over the dedicated link serving client site $k$ whenever: (1) the multicast is triggered by a client request from client site $k$ (with probability $\lambda_k/\lambda$; or (2) the multicast is triggered by a client request from one of the other sites, but at least one client from client site $k$ requests the file during the period of length $x + d$ from when the triggering request was made until the time of the multicast (with probability $\frac{\lambda - \lambda_k}{\lambda}\left(1 - e^{-\lambda_k(x+d)}\right)$ assuming Poisson arrivals).

The upper bound on the minimum required network bandwidth for the shared link with fan-out K topology thus can be expressed as:

$$B_{min}^{net} < \frac{1}{2}\left(\ln\left(\frac{N}{ND+1}+1\right) + \sum_{k=1}^{K}\int_0^T \frac{\frac{\lambda_k}{\lambda} + \frac{\lambda-\lambda_k}{\lambda}\left(1-e^{-\lambda_k(x+d)}\right)}{x+d+1/\lambda}dx\right), \qquad (3.2)$$

which can also be written as:

$$B_{min}^{net} < \frac{1}{2}\left((K+1)\ln\left(\frac{N}{ND+1}+1\right) - \sum_{k=1}^{K}\frac{\lambda-\lambda_k}{\lambda}\int_0^T \frac{e^{-\lambda_k(x+d)}}{x+d+1/\lambda}dx\right). \qquad (3.3)$$

For fixed request rate per client site (i.e., fixed $N_k$ and bounded relative differences among the $N_k$), and fixed client delay $D$, the second term (inside the outer parentheses) on the right-hand side of relation (3.2) dominates the first as $K$ is scaled up. This term, and thus the upper bound, scales as:

$$\sum_{k=1}^{K}\int_0^T \frac{1-e^{-\lambda_k(x+d)}}{x+d+1/\lambda}dx,$$

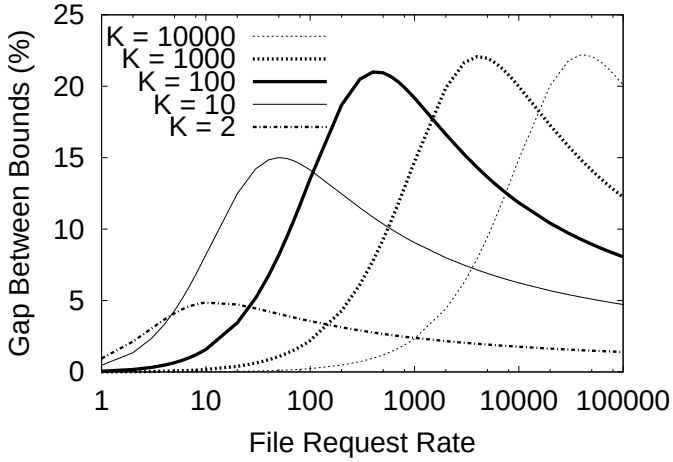
which is $\Theta(K)$, the same as the lower bound. (For $d > 0$, the integral is bounded between $\int_0^T \frac{1-e^{-\lambda_k(x+d)}}{x+d+1/\lambda_k}dx$ and $T/d$, while for $d = 0$, the integral is bounded between $\int_0^T \frac{1-e^{-\lambda_k x}}{x+1/\lambda_k}dx$ and, since $\frac{1-e^{-\lambda_k x}}{x} < \lambda_k$ for $x > 0$, $\lambda_k T$.) This topology (as well as the topology obtained by omitting the shared link) represents a worst case with respect to how network bandwidth scales with the number of client sites.

For fixed number of client sites $K$, the first term on the right-hand side of relation (3.3) dominates the second as $N$ is scaled up or $D$ is scaled down, and thus the upper bound, like the lower bound, is $\Theta\left(\ln\frac{N}{ND+1}\right)$.

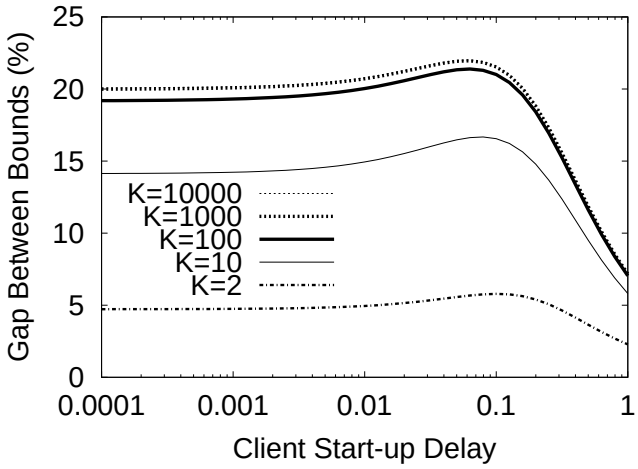
Figure 3.2 plots the percent difference between the bounds given in relations (3.1) and (3.3)[1], relative to the lower bound, as a function of the normalized file request rate $N$ for immediate service ($D = 0$), and as a function of the client delay $D$, for the case of equal request rate at each client site ($N_k = N/K$ for all $k$). As can

[1]In the case of relation (3.3) and others where a closed form has not been derived, numerical results are obtained using Maple [119].

be seen, the upper and lower bounds on minimum required network bandwidth are quite tight, particularly for $K < 10$. The bounds are similarly tight for $0 \leq D \leq 0.1$, and are much tighter for $D > 0.1$.




(a) $D = 0$




(b) $N_k = 10$ for all $k$

Figure 3.2: Tightness of Bounds for Shared Link with Fan-out $K$ Topology

### 3.2.3 Daisy-chain Topology

The previous section derived a lower and an upper bound on the minimum required network bandwidth for a multicast delivery tree topology where all but one tree link is shared as little as possible. This section considers a daisy-chain topology (as shown in Figure 3.3) where the tree links are shared as much as possible. In this

topology, the total client request rate on the link that connects client sites $k$ and $k+1$ is equal to the sum of the request rates from the k client sites served by that link, that is, $\sum_{i=1}^{k} \lambda_i$.

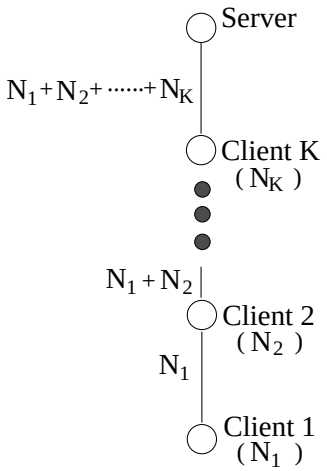



Figure 3.3: Daisy-chain Topology

A lower bound on the minimum required network bandwidth for on-demand streaming on this topology can be derived in an analogous fashion as for the shared link with fan-out $K$ topology, yielding:

$$B_{min}^{net} > \frac{2}{K+1} \sum_{k=1}^{K} \ln \left( \frac{\sum_{i=1}^{k} N_i}{D \sum_{i=1}^{k} N_i + 1} + 1 \right). \tag{3.4}$$

where the average end-to-end unit bandwidth cost is equal to $\frac{\sum_{i=1}^{K} i}{K} = \frac{K+1}{2}$ times the unit bandwidth cost on each link.

For immediate service (i.e., $D = 0$) and fixed request rate at each client site, as $K$ increases relation (3.5) scales as:

$$\frac{1}{K} \sum_{k=1}^{K} \ln k = \frac{1}{K} \ln(K!).$$

According to Stirling's Approximation for $K!$, the right-hand side can be approximated as:

$$\frac{1}{K}\ln\left(\sqrt{2\pi K}(K/e)^K\right),$$

which is $\Theta(\ln K)$. On the other hand, for fixed $D > 0$ and fixed $N_k$, relation (3.4) is $\Theta(1)$ with respect to scaling with $K$. For a fixed number of client sites $K$ and varying $N$ or $D$, relation 3.4 is $\Theta\left(\ln\frac{N}{ND+1}\right)$.

An upper bound on the minimum required network bandwidth for the daisy-chain topology can be obtained in a similar manner as for the shared link with fan-out $K$ topology, yielding:

$$B_{min}^{net} < \frac{2}{K+1}\left(K\ln\left(\frac{N}{ND+1}+1\right) - \sum_{k=1}^{K-1}\frac{\lambda-\sum_{i=1}^{k}\lambda_i}{\lambda}\int_0^T \frac{e^{-(x+d)\sum_{i=1}^{k}\lambda_i}}{x+d+1/\lambda}dx\right). \quad (3.5)$$

The first term (inside the outer parentheses) on the right-hand side of relation (3.5) dominates the second. Thus, for immediate service and fixed request rate at each client site, the upper bound in relation (3.5) is $\Theta(\ln K)$. For $D > 0$, on the other hand, the scaling with $K$ is, again, $\Theta(1)$. For fixed number of client sites $K$ and varying $N$ or $D$, the upper bound in relation (3.5) is $\Theta\left(\ln\frac{N}{ND+1}\right)$.

The daisy-chain topology represents a best case with respect to how network bandwidth scales with the number of client sites, since from equation (2.2) network bandwidth must be at least $\Theta\left(\ln\frac{N}{ND+1}\right)$, or at least $\Theta(\ln K)$ when $D = 0$ since $N$ scales linearly with $K$, or $\Theta(1)$ for $D > 0$.

Figure 3.4 graphs the percent difference between the bounds given in relations (3.4) and (3.5), relative to the lower bound, for the case of equal request rates per client site ($N_k = N/K$ for all $k$) and immediate service. The bounds are very tight with the percent difference under 7% in all cases. Although not presented, the bounds are similarly tight or tighter for $D > 0$.

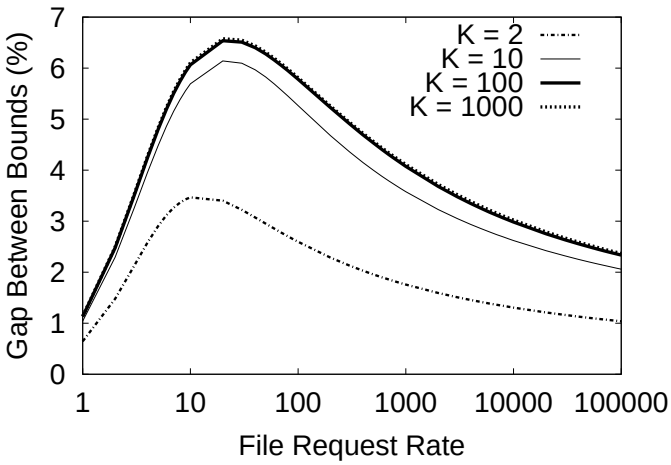



Figure 3.4: Tightness of Bounds for Daisy-chain Topology ($D = 0$)

### 3.2.4 Balanced Tree Topology

Previous sections considered two multicast delivery tree topologies that capture the extreme points with respect to the scaling of the network bandwidth requirement with the number of client sites. This section considers a balanced tree topology that yields intermediate behaviour, in which all nodes but the root correspond to client sites. Specifically, Figure 3.5 depicts a balanced binary tree of depth $L$ with the number of client sites $K$ equal to $2^{L+1} - 2$. Although this special type of tree is assumed in the following analyses, the results are qualitively similar for balanced mary trees, and for trees in which only leaf nodes correspond to client sites. Because of the complexity of this topology, the following analyses assume an identical client request rate at each client site. The same scaling behaviour of network bandwidth is obtained if the request rates at different client sites have bounded relative differences.

A lower bound on the minimum required network bandwidth for on-demand streaming on this topology, assuming equal request rate at each client site, can be derived in a similar fashion as for the previous topologies, yielding:

$$B_{min}^{net} > \frac{2^L - 1}{2^L(L-1)+1} \sum_{i=1}^{L} 2^i \ln\left(\frac{\frac{2^{L-i+1}-1}{K}N}{\frac{2^{L-i+1}-1}{K}ND + 1} + 1\right). \tag{3.6}$$

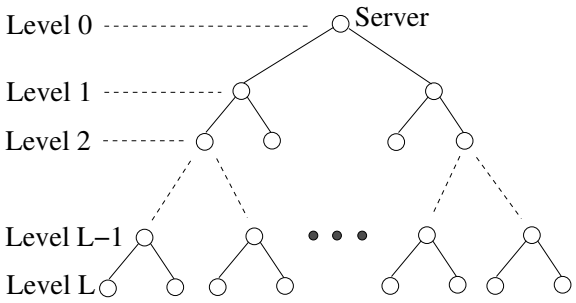



Figure 3.5: Balanced Tree Topology

In the above expression, the average end-to-end unit bandwidth cost is equal to $\frac{\sum_{i=1}^{L} i \cdot 2^i}{K} = \frac{2^L(L-1)+1}{2^L-1}$ times the unit bandwidth cost on each link.

For fixed request rate at each client site (i.e., fixed $N/K$) and fixed maximum client delay $D$, for large $K$ (and $L$), the above bound is approximately:

$$\frac{1}{L}\sum_{i=1}^{L} 2^i \ln\left(\frac{\frac{2^{L-i+1}-1}{K}N}{\frac{2^{L-i+1}-1}{K}ND+1}+1\right).$$

For $D > 0$, the log factors are bounded between $\ln\left(\frac{N/K}{ND/K+1}+1\right)$ and $\ln\left(\frac{1}{D}+1\right)$, and thus the bound is $\Theta\left(\frac{1}{L}\sum_{i=1}^{L} 2^i\right)$, which is $\Theta\left(K/\ln K\right)$. For $D = 0$, the above expression is approximately:

$$\begin{aligned}
&\frac{1}{L}\sum_{i=1}^{L} 2^i \ln\left(2^{L-i+1}\frac{N}{K}\right) \\
=&\frac{1}{L}\sum_{i=1}^{L} 2^i \left((L-i+1)\ln 2 + \ln\frac{N}{K}\right) \\
=&\frac{1}{L}\left(2^{L+2}-2^2-2L\right)\ln 2 + \frac{1}{L}\left(2^{L+1}-2\right)\ln\frac{N}{K},
\end{aligned}$$

which is, again, $\Theta\left(K/\ln K\right)$.

On the other hand, with a fixed number of client sites $K$ (and thus fixed $L$) and varying $N$ or $D$, the bound in relation (3.6) is $\Theta\left(\ln\frac{N}{ND+1}\right)$.

An upper bound on the minimum required network bandwidth for the balanced binary tree topology can be derived analogously as for the previous topologies, yielding:

$$B_{min}^{net} < \frac{2^L - 1}{2^L(L-1)+1} \sum_{i=1}^{L} 2^i \int_0^T \frac{1}{x+d+1/\lambda} \left( \frac{2^{L-i+1}-1}{K} + \frac{K-(2^{L-i+1}-1)}{K} \left(1 - e^{-(x+d)\frac{\lambda}{K}(2^{L-i+1}-1)}\right)\right) dx, \tag{3.7}$$

which can also be written as:

$$B_{min}^{net} < \frac{2^L - 1}{2^L(L-1)+1} \left( K \ln\left(\frac{N}{ND+1}+1\right) - \sum_{i=1}^{L} 2^i \frac{K-(2^{L-i+1}-1)}{K} \int_0^T \frac{e^{-(x+d)\frac{2^{L-i+1}-1}{K}\lambda}}{x+d+1/\lambda} dx \right). \tag{3.8}$$

For fixed request rate at each client site (i.e., fixed $\lambda/K$ and $N/K$) and fixed maximum client delay $D$, for large $K$ (and $L$), the bound in relation (3.7) is approximately:

$$\frac{1}{L} \sum_{i=1}^{L} 2^i \int_0^T \frac{1}{x+d+1/\lambda} \left( \frac{2^{L-i}}{K} + \frac{K-2^{L-i}}{K} \left(1 - e^{-(x+d)\frac{\lambda}{K}2^{L-i}}\right)\right) dx.$$

This expression scales as:

$$\int_0^T \frac{1}{x+d+1/\lambda} dx + \frac{1}{L} \sum_{i=1}^{L} 2^i \frac{K-2^{L-i}}{K} \int_0^T \frac{1}{x+d+1/\lambda} \left(1 - e^{-(x+d)\frac{\lambda}{K}2^{L-i}}\right) dx.$$

For $d > 0$, the first term is $\Theta(1)$, and the second term is $\Theta(2^L/L)$ or $\Theta(K/\ln K)$ since the integral is bounded by $\int_0^T \frac{1-e^{-(x+d)\lambda/K}}{x+d+1/(\lambda/K)} dx$ and $T/d$. For $d = 0$, the first term is $\Theta(\ln K)$, and for large $K$ (and $L$), the second term is approximately:

$$\frac{1}{L} \sum_{i=1}^{L} 2^i \left( \int_0^{T/2^{L-i}} \frac{1-e^{-x\frac{\lambda}{K}2^{L-i}}}{x} dx + \int_{T/2^{L-i}}^{T} \frac{1-e^{-x\frac{\lambda}{K}2^{L-i}}}{x} dx \right).$$

For the first integral, since the derivative of the integrand with respect to $x$ is negative, the integral is bounded from above by:

$$\left( \frac{1 - e^{-x\frac{\lambda}{K}2^{L-i}}}{x} \bigg|_{x=0} \right) \frac{T}{2^{L-i}} = \frac{\lambda T}{K} = N/K.$$

For the second integral, if the numerator of the integrand is replaced by 1, the integral can be evaluated as $(L-i)\ln 2$. The whole expression is thus bounded from above by:

$$\frac{1}{L} \sum_{i=1}^{L} 2^i \left( (L-i)\ln 2 + \frac{N}{K} \right),$$

which is $\Theta(2^L/L)$ or $\Theta(K/\ln K)$. Finally, considering again the first integral and using the fact that the derivative of the integrand is negative, the whole expression can be seen to be bounded from below by:

$$\frac{1}{L} \sum_{i=1}^{L} 2^i \left( 1 - e^{-(N/K)} \right),$$

which is also $\Theta(2^L/L)$ or $\Theta(K/\ln K)$. Therefore, as $K$ is scaled up, the upper bound of relation (3.7) is $\Theta(K/\ln K)$. On the other hand, for a fixed number of client sites $K$ (and fixed $L$) and varying $N$ or $D$, from relation (3.8) the upper bound is $\Theta\left(\ln \frac{N}{ND+1}\right)$. Results for randomly generated network topologies and client site locations, as presented in the next section, suggest that this scaling behavior is representative of what might be expected in practice.

Figure 3.6 illustrates that for immediate service (i.e., $D = 0$), the bounds given in relations (3.6) and (3.8) are quite tight, with percent difference relative to the lower bound under 20% for all system configurations.

Table 3.2 summarizes the scaling of the minimum required network bandwidth for on-demand streaming on the three canonical multicast delivery trees, in comparison to that of unicast delivery.

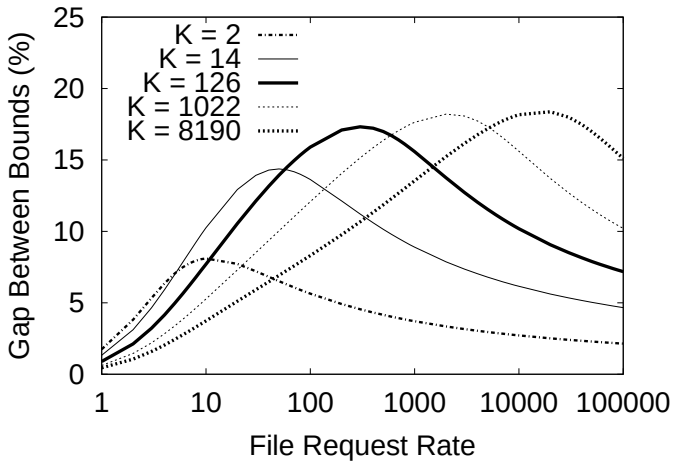



Figure 3.6: Tightness of Bounds for Balanced Tree Topology ($D = 0$)

Table 3.2: Scaling of Network Bandwidth for Multicast

| Topology | with $K$ | | with $N$ or $D$ |
|---|---|---|---|
| | $D = 0$ | $D > 0$ | |
| Shared Link with Fan-out $K$ | $\Theta(K)$ | $\Theta(K)$ | $\Theta\left(\ln \frac{N}{ND+1}\right)$ |
| Daisy-Chain | $\Theta(\ln K)$ | $\Theta(1)$ | $\Theta\left(\ln \frac{N}{ND+1}\right)$ |
| Balanced Tree | $\Theta\left(K/\ln K\right)$ | $\Theta\left(K/\ln K\right)$ | $\Theta\left(\ln \frac{N}{ND+1}\right)$ |
| Unicast | $\Theta(K)$ | $\Theta(K)$ | $\Theta(N)$ |

## 3.3 Multicast Delivery in Large Networks

In Section 3.2, a wide range of behaviours was observed with respect to the scaling of the minimum required network bandwidth for on-demand streaming with the number of client sites. This section considers the question of what type of scaling behaviour is likely to be seen in practice.

A related issue concerns the topologies of multicast delivery trees. The results from Section 3.2 show that daisy-chain networks can achieve substantially lower network bandwidth than fan-out networks. However, these results are based on the assumption that for the multicast delivery tree under consideration, the network distance from the server to a random client has constant expected value as the

number of client sites scales up. In practice, it may be difficult to realize this property in daisy-chain networks because of the disparate locations of client sites. Therefore, there is a trade-off between sharing as many links as possible, and minimizing the network distance from the server to each client. It is not clear *a priori* whether alternative multicast delivery tree structures can achieve significantly lower network bandwidth usage than the commonly used shortest path trees, in the context of multicast-based on-demand streaming.

This section addresses the above questions through experiments on networks with randomly generated topologies and client site locations. Section 3.3.1 describes the topology generators used for constructing the networks. Section 3.3.2 discusses five algorithms for building multicast delivery trees. Section 3.3.3 compares the network bandwidth requirements of the multicast delivery trees established by these algorithms, and examines the scaling of network bandwidth with the number of client sites.

### 3.3.1 Network Topology Generators

In this study, two network topology generators GT-ITM [21, 128] and Inet-2.1 [63] are used to generate the underlying network topologies.

GT-ITM (Georgia Tech Internetwork Topology Models) is an internetwork topology generator developed at the Georgia Institute of Technology. It is a collection of routines that generate networks based on a variety of graph models. In a graph model, a node represents a switch or a router, and an edge represents a direct connection between two switches or routers. In particular, this study uses two flat random graph models, the Pure Random model and the Waxman model [120], and one hierarchical graph model, the Transit-Stub (TS) model [128]. In the two random graph models, nodes are distributed at random on a plane. In the Pure Random model, an edge is added between a pair of nodes with probability $p$. In the Waxman model, the probability with which an edge is added from a node $u$ to another node $v$ is given by $\alpha e^{-d(u,v)/(\beta L)}$, where $0 < \alpha$, $\beta \leq 1$ are input parameters, $d(u, v)$ is the

Euclidean distance from $u$ to $v$, and $L$ is the maximum distance between any two nodes of the graph. A larger value of $\alpha$ increases the ratio of the number of long edges to short edges, and a bigger $\beta$ results in a larger average node degree. For the network topologies generated in this study, $\alpha = 0.3$ and $\beta = 0.15$ were used.

In the Transit-Stub model, a network is viewed as a collection of interconnected routing domains, each of which can be further classified as either a *transit* domain or a *stub* domain. A domain is a stub domain if the network path connecting any two nodes $u$ and $v$ goes through that domain only if either $u$ or $v$ (or both) is in that domain. A transit domain is used to interconnect stub domains together, and therefore does not have this restriction. A transit-stub network is generated by first constructing a connected random graph using any of the graph models defined inside GT-ITM. Each node in that graph represents an entire transit domain. Then each node is replaced by another connected random graph, representing the backbone topology of the respective transit domain. Finally, for each node in each transit domain, a number of connected random graphs representing the stub domains are generated and attached to that node.

Inet is an Internet autonomous system (AS) topology generator developed at the University of Michigan. In the Internet, an AS system is a network under a single administrative authority. ASs connect with each other through border routers. Within each AS, the network could be further partitioned into subnetworks connected by internal routers. Rather than modeling the Internet as a network of interconnected routers (e.g., as GT-ITM does), Inet models it as a network of interconnected ASs based on the three power laws described in [40]. In this study, the network topologies were generated using Inet version 2.1.

### 3.3.2 Tree Construction Algorithms

In this section, five algorithms for building multicast delivery trees are considered. These algorithms are compared in terms of the network bandwidth requirement of the resulting multicast delivery trees. However, the issue of how these algorithms

can be really implemented is not considered, since the objective of this study is only to understand the potential network bandwidth savings with alternative multicast delivery tree structures. Moreover, application-level multicast may provide considerable flexibility in how multicast delivery trees are constructed. Note that some of the algorithms considered below do not restrict a delivery tree to branch only at the client/server sites, as could be feasible if, for example, this tree were built using network-level multicast or on top of an application-level multicast infrastructure [23].

The five algorithms considered in this study are as follows:

- **SP** (Shortest Path): A shortest path tree is constructed by merging the shortest paths from the server to each client site. If two shortest paths share a common link, only one copy of the data will be transmitted along that link. Note that in the topologies considered in this study, network links and paths are symmetric, and therefore shortest path trees are the same as the reverse shortest path trees that are built by common Internet multicast routing algorithms. The shortest path trees are used as a baseline against which alternative delivery tree structures are compared.

- **GL-A** (Greedy Link-All): This algorithm attempts to minimize the number of links in the resulting delivery tree. This algorithm builds a delivery tree as follows. Initially, the tree contains only the server node. Then the client sites are added incrementally into the tree. In each step, the shortest paths from all nodes in the delivery tree (including non-client/server nodes) to every client site not belonging to the tree are computed, and the shortest path with the fewest number of links not already in the delivery tree is selected. (If more than one such path exists, one of them is randomly chosen.) The client site and the new links on that path are added into the current delivery tree.

- **GL-P** (Greedy Link-Participants): This algorithm also aims at minimizing the total number of links in the resulting delivery tree. Like GL-A, this algorithm adds client sites incrementally into the delivery tree. However, in each step, only the shortest paths originating at the client/server sites in the

delivery tree are considered. In comparison to GL-A, this algorithm has a lower computational cost for determining which client to add next and the corresponding attachment point.

- **GC-A** (Greedy Cost-All): This algorithm attempts to minimize the total network bandwidth cost of the resulting delivery tree. It works similarly to GL-A, except that in each step, the shortest path with the lowest incremental network bandwidth cost (if this path were merged into the current delivery tree), rather than with the fewest number of new links, is selected. The network bandwidth cost of a delivery tree is computed using the analysis technique developed in Section 3.2 for the lower bound on the minimum required network bandwidth. The upper bound analysis could also be applied, but the lower bound analysis has lower computational cost, and as shown in Section 3.2, the difference between the two bounds is small. Compared to the algorithms described above, GC-A has the disadvantage that the request rate at each client site has to be known *a priori*. However, the multicast delivery trees generated by this algorithm may be able to achieve a better compromise between sharing as many links as possible, and minimizing the network distance over the links of the tree between the server and each client.

- **GC-P** (Greedy Cost-Participants): This algorithm is analogous to GL-P, except that it considers the incremental network bandwidth cost when choosing the client site to add next. The main advantage of this algorithm in comparison to GC-A is its relatively low complexity.

### 3.3.3 Network Bandwidth Results

The experiments conducted here use two network sizes: 600 nodes (GT-ITM), and 6000 nodes (Inet-2.1). 6000 nodes is a typical network size for Internet AS-level topology experiments, and also is sufficiently large to permit use of power law rules in topology generator [63]. For each network size, ten networks are generated using different random number seeds. For the ten 600-node networks, five are generated

using the Waxman model and the others using the Transit-Stub model (Pure Random model is used to generate the sub-network inside each Transit or Stub domain). For each network generated, the server site is first randomly chosen from the nodes having the maximum degree. The client sites are then randomly selected from the remaining nodes. The experiments use three choices for the fraction of nodes that are client sites: 2%, 10%, and 50% of the total number of nodes. The request rates at the client sites are assumed to be identical. Three per-site request rates are considered: $N_k = 1$, $N_k = 10$, and $N_k = 100$. After these parameters are chosen, the five algorithms discussed in Section 3.3.2 are applied to generate multicast delivery trees. For each delivery tree, the lower bound on the minimum required network bandwidth is computed by summing the minimum required network bandwidth on each tree link.

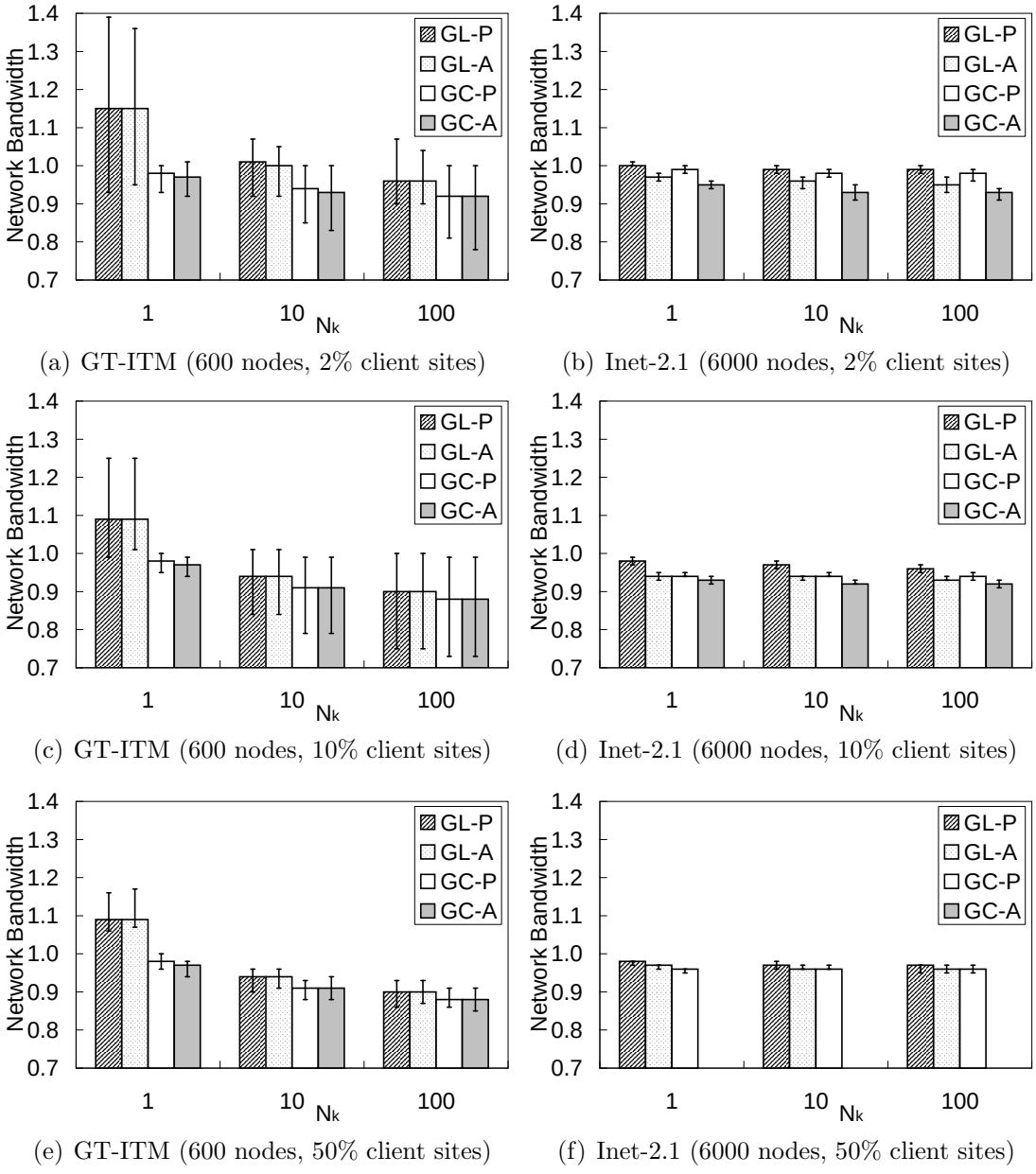
Table 3.3 summarizes the experimental parameters and their values used in the experiments. These values are chosen not because they are necessarily representative for real large internetworks such as the Internet, but to achieve greater confidence in the validity of the experimental results through consideration of a wide range of environments.

Table 3.3: Experimental Parameters

<table>
<tr><th colspan="2">Network Topology</th><th># of nodes</th><th colspan="3"># of clients</th><th>Server</th><th colspan="3">$N_k$</th></tr>
<tr><td rowspan="2">GT-ITM</td><td>Waxman<br>Transit-Stub</td><td>600</td><td rowspan="2">2%</td><td rowspan="2">10%</td><td rowspan="2">50%</td><td rowspan="2">Maximum<br>Connectivity</td><td rowspan="2">1</td><td rowspan="2">10</td><td rowspan="2">100</td></tr>
<tr><td colspan="2">Inet-2.1</td><td>6000</td></tr>
</table>

#### 3.3.3.1 Comparison of Tree Construction Algorithms

Figure 3.7 compares the network bandwidth requirements of multicast delivery trees generated by the algorithms described in Section 3.3.2. In the figure, each bar shows the minimum, average, and maximum value of the minimum required network bandwidth for on-demand streaming with immediate service (i.e., $D = 0$) over




(a) GT-ITM (600 nodes, 2% client sites)

(b) Inet-2.1 (6000 nodes, 2% client sites)

(c) GT-ITM (600 nodes, 10% client sites)

(d) Inet-2.1 (6000 nodes, 10% client sites)

(e) GT-ITM (600 nodes, 50% client sites)

(f) Inet-2.1 (6000 nodes, 50% client sites)

Figure 3.7: Network Bandwidth for Alternative Delivery Trees
(Each bar shows the minimum, average, and maximum required network bandwidth, relative to the corresponding bandwidths with SP, over ten randomly-generated network topologies)

ten multicast delivery trees constructed by the indicated algorithm, normalized to the corresponding bandwidths with SP. (Results for $D > 0$ show similar trends.) The required network bandwidth is computed by applying the lower bound analysis techniques for the minimum required network bandwidth from Section 3.2. Applying the upper bound analysis instead yields similar trends. For the 6000-node networks

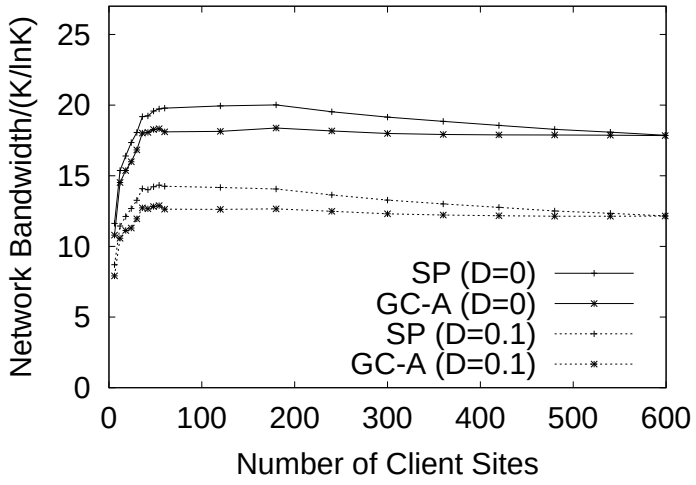
with 50% of the nodes client sites, results for GC-A were not obtained owing to the prohibitively high computational cost.

The main conclusions drawn from Figure 3.7 are:

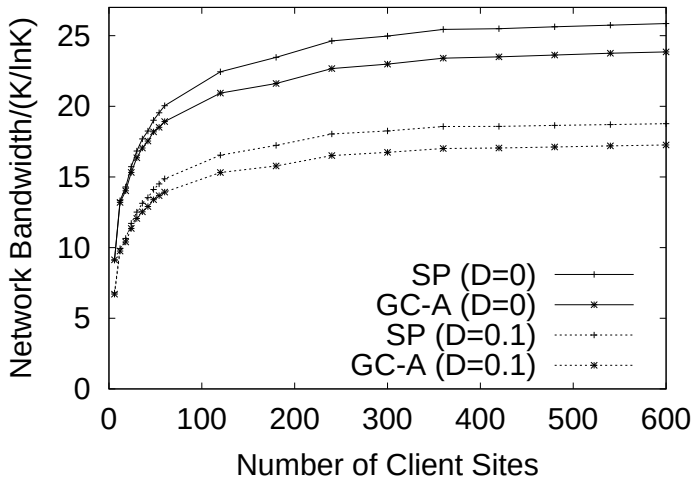
- GC-A performs the best. This is not surprising, as it takes into account the request rate at each client site to determine the incremental network bandwidth cost of each potential addition to the delivery tree.
- Shortest path trees (as constructed by SP) have consistently similar required network bandwidth as those constructed by GC-A for low request rates, and are typically only modestly more expensive for higher request rates.
- The protocols GL-A, GL-P and GC-P yield only marginally higher required network bandwidth than GC-A, in most cases. However, for relatively low request rates and few client sites, GL-A and GL-P can perform poorly.

#### 3.3.3.2 Scaling of Minimum Network Bandwidth

Figure 3.8 illustrates how the network bandwidth requirements of multicast delivery trees generated by algorithms described in Section 3.3.2 scale with the number of client sites, for both immediate service ($D = 0$) and bounded-delay service ($D = 0.1$). In the figure, each data point shows the average value over ten different networks. The average value is presented because the difference among the results across those networks is very small (within 1%). The normalized per-site request rate is fixed at 10. Since the analyses in Section 3.2 suggest that the minimum network bandwidth scales as $\Theta(K/\ln K)$ in large networks, the figure depicts the ratio of the network bandwidth usage to $K/\ln K$. As shown in the figure, the minimum network bandwidth grows more quickly when the number of client sites is small, since there is not much sharing of tree links. For large number of client sites ($K$ greater than 5% of the number of network nodes), the minimum network bandwidth appears to be $\Theta(K/\ln K)$, as indicated by the flattening of the curves.




(a) GT-ITM Topologies (600 nodes)




(b) Inet-2.1 Topologies (6000 nodes)

Figure 3.8: Scaling of Minimum Network Bandwidth
($N_k = 10$ for all $k$)

## 3.4 Network Bandwidth of Hierarchical Stream Merging

This section addresses the following two questions: (1) can practical immediate service protocols achieve close to the minimum required network bandwidth? and (2) is it possible to achieve close to the minimum network and server bandwidth usage simultaneously? Specifically, the hierarchical stream merging protocol and its variants [36, 38] are considered, since they can achieve reasonably close to the minimum required server bandwidth [38].

This section is organized as follows. Section 3.4.1 discusses the network bandwidth efficiency of *network-naïve* hierarchical stream merging, i.e., hierarchical stream merging as described in [36, 38]. Section 3.4.2 proposes *network-aware* hierarchical stream merging, a variant that attempts to further reduce network bandwidth usage by taking into account client locations when aggregating clients into multicast groups, and compares the server and network bandwidth usage of network-aware hierarchical stream merging against the lower bounds and that of network-naïve hierarchical stream merging.

### 3.4.1 Network-naïve Hierarchical Stream Merging

The variant of hierarchical stream merging considered here requires that a client receive bandwidth of twice the file playback bit rate, while the network bandwidth bounds derived in Section 3.2 assume unlimited client receive bandwidth. In the following, these bounds are heuristically extended for constrained client reception rate, for the case of immediate service.

As presented in Section 2.4, a conjectured asymptotic lower bound on the required *server* bandwidth for on-demand streaming with immediate service and clients that can receive two playback bit rate streams concurrently is given by [38]:

$$1.62 \ln(\frac{N}{1.62} + 1). \tag{3.9}$$

The lower bounds on the minimum required network bandwidth in Section 3.2 are derived by summing simply the minimum network bandwidth requirement on each tree link, which is computed using the same formula as for the minimum required server bandwidth (expression (2.3)). This network bandwidth can be heuristically modified for the case of client receive bandwidth equal to twice the file playback bit rate, by using expression (3.9) instead of expression (2.3).

For the shared link with fan-out $K$ topology (shown in Figure 3.1), and immediate service, this yields:

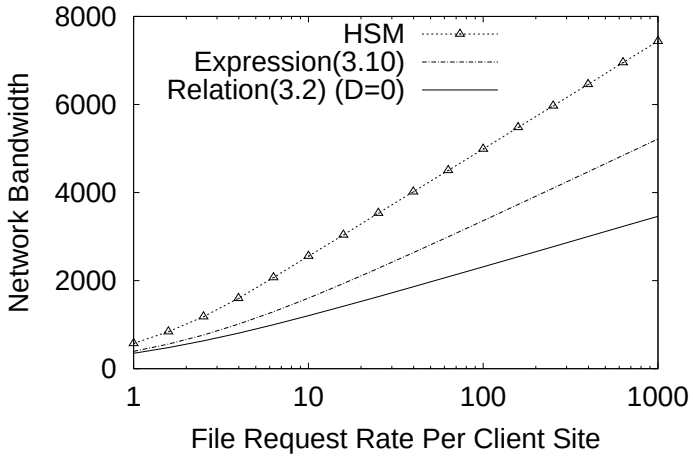
$$\frac{1}{2}\left(1.62\ln(\frac{N}{1.62}+1)+\sum_{k=1}^{K}1.62\ln(\frac{N_k}{1.62}+1)\right). \tag{3.10}$$

The accuracy of this expression could be evaluated by running simulations to obtain lower and upper bounds on the minimum network bandwidth usage. A lower bound can be obtained by simulating the optimal immediate service policy [38] on each tree link and summing the bandwidth requirements together. To obtain an upper bound, delivery on the complete fan-out $K$ topology can be simulated, using the policy that is optimal for (only) the shared link. This evaluation is left for future work.

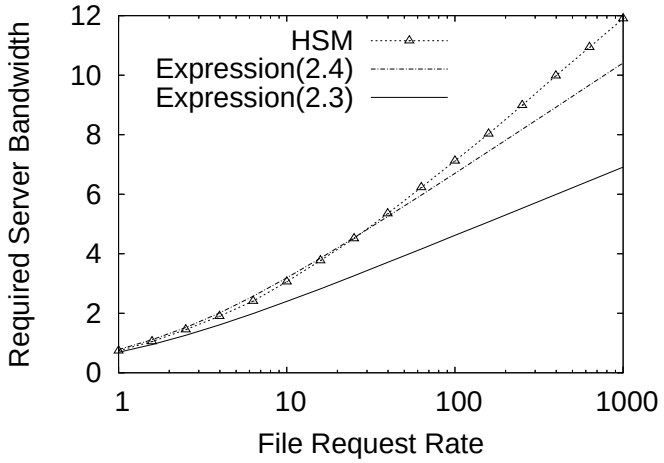
Figure 3.9(a) compares the network bandwidth usage of network-naïve hierarchical stream merging (specifically, the CT variant), as obtained from simulation[2], against the lower bound of relation (3.1) (with $D = 0$), and expression (3.10), for $K = 1000$ and equal request rate at each client site, as functions of the normalized per-site request rate $N_k$. The gap between the hierarchical stream merging results and expression (3.10) (and qualitatively similar results for other topologies) suggests that there may be only modest room for improvement over network-naïve hierarchical stream merging with respect to network bandwidth usage.

Figure 3.9(b) shows the server bandwidth requirement of network-naïve hierarchical stream merging (the CT variant), the conjectured asymptotic lower bound on server bandwidth from expression (2.4) for the case of clients that can receive two playback bit rate streams concurrently, and the lower bound on required server bandwidth from equation (2.3). The reasonably small gaps between the required bandwidth with hierarchical stream merging and the lower bounds in both figures suggest that it is indeed possible to achieve reasonably close to the minimal network and server bandwidth usage simultaneously, with a practical immediate service on-demand streaming protocol.

[2]All simulation results have 95% confidence intervals that are within 5% of the reported values. Poisson request arrivals are assumed.




(a) Network Bandwidth (Figure 3.1 shared link with fan-out $K$ topology, $K = 1000$)




(b) Server Bandwidth

Figure 3.9: Bandwidth Efficiencies of Hierarchical Stream Merging

### 3.4.2 Network-aware Hierarchical Stream Merging

Although the gap shown in Figure 3.9(a) between the network bandwidth requirement of network-naïve hierarchical stream merging and the network bandwidth from expression (3.10) is not large, it is greater than the gap shown in Figure 3.9(b) between the server bandwidth usage of hierarchical stream merging and expression (2.4). This section explores the potential for further reducing network bandwidth usage from use of a *network-aware* hierarchical stream merging policy, which merges clients only if they are from the same site.

In order to gain good insights on the network bandwidth efficiency of network-aware hierarchical stream merging, a delivery tree topology in which clients from different sites do not share any tree links is considered (shown in Figure 3.10). Since there are no shared links, no benefit is gained from merging clients from different sites. Therefore, the improvement in network bandwidth usage through use of network-aware hierarchical stream merging should be maximized.

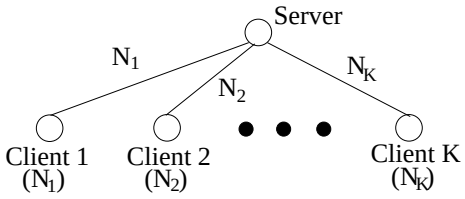
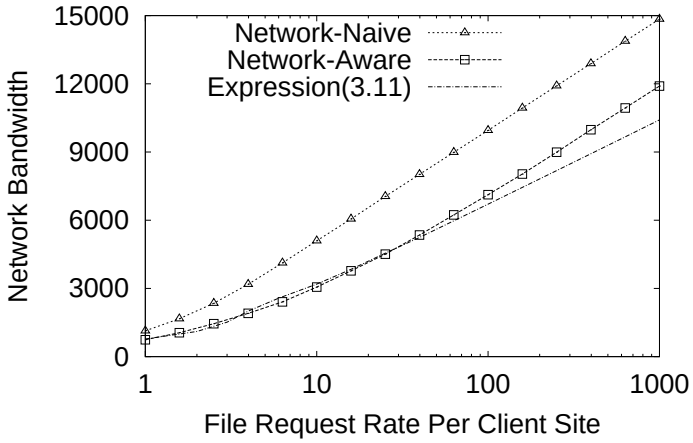



Figure 3.10: Fan-out Topology With No Shared Links

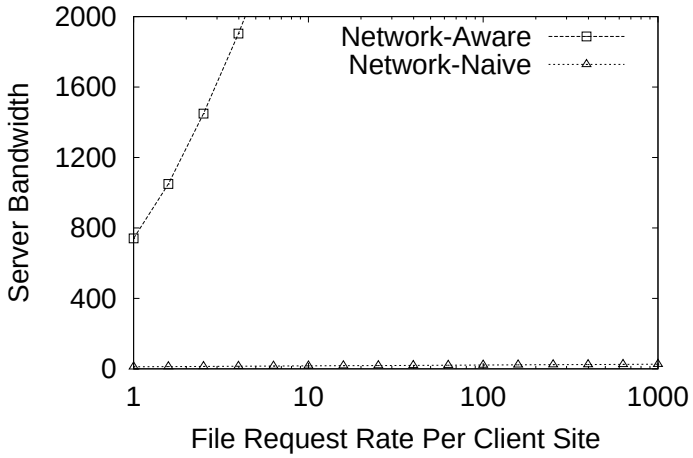
For the delivery tree topology shown in Figure 3.10, an approximation for the minimum required network bandwidth for on-demand streaming with immediate service for the case of clients that can receive two playback bit rate streams concurrently, can be derived in a similar fashion as expression (3.10), yielding:

$$\sum_{k=1}^{K} 1.62 \ln(\frac{N_k}{1.62} + 1). \tag{3.11}$$

Figure 3.11(a) compares the required network bandwidth of network-aware and network-naïve hierarchical stream merging, and expression (3.11), for the topology shown in Figure 3.10 with $K = 1000$, as a function of the normalized per-site client request rate $N_k$ assuming equal request rate at each client site. As shown in the figure, network-aware hierarchical stream merging can reduce the required network bandwidth to very close to that from expression (3.11). However, as shown in Figure 3.11(b), the cost of this saving in network bandwidth usage is a three orders of magnitude increase in the required server bandwidth. Moreover, aggregating clients from the same site can have a negative impact on the network bandwidth usage of shared links.




(a) Required Network Bandwidth




(b) Required Server Bandwidth

Figure 3.11: Network-aware vs. Network-naïve Hierarchical Stream Merging
(Figure 3.10 fan-out topology, $K = 1000$)

To illustrate this impact, Figure 3.12 depicts the network bandwidth requirements of network-aware and network-naïve hierarchical stream merging for the shared link with fan-out $K$ topology shown in Figure 3.1. As seen in the figure, with the presence of an additional shared link, network-naïve hierarchical stream merging consumes less network bandwidth than network-aware hierarchical stream merging. Figure 3.11(b) and Figure 3.12 suggest that substantial reduction in network bandwidth usage from that of network-naïve hierarchical stream merging may be infeasible in real environments.

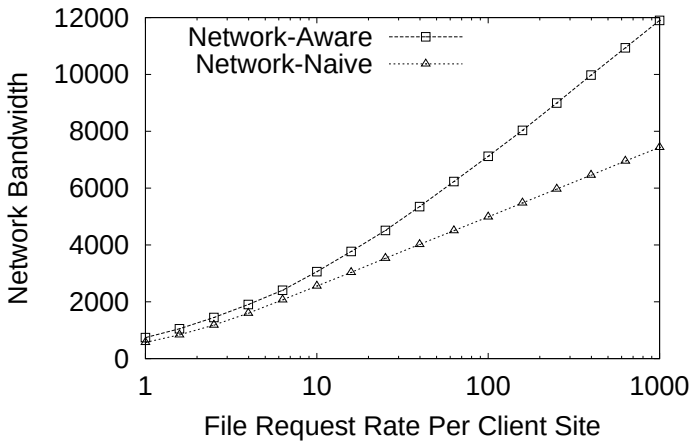



Figure 3.12: Network-aware vs. Network-naïve Polices with Shared Link
(Figure 3.1 topology, $K = 1000$)

## 3.5 Network Bandwidth of Periodic Broadcast

This section addresses the network bandwidth requirements of practical periodic broadcast protocols. A new variant of the optimized periodic broadcast (OPB) protocol [76], OPB/tou, is proposed in Section 3.5.1. The results from that section show that unlike previously proposed periodic broadcast protocols, OPB/tou can achieve reasonably close to the minimum required server bandwidth usage even under light load. The network bandwidth efficiency of OPB/tou is then studied in Section 3.5.2 by comparing its network bandwidth usage with the lower bounds derived in Section 3.2, and that of network-naïve hierarchical stream merging. Table 3.4 summarizes the notation used in the following presentation.

### 3.5.1 OPB/tou

In the OPB protocol, a file is partitioned into $M$ segments, each of which is repeatedly broadcast on a separate channel at a rate $r$. The segment size progression is chosen such that if clients initially listen on the channels 1 to $s$ that deliver the first $s$ segments, switch from the channel transmitting segment $m$ to the channel transmitting segment $m + s$ $(m + s \leq M)$ after segment $m$ is fully received, and begin the playback of the file after the first segment is downloaded in its entirety,

every segment can be received just in time for playback. The server bandwidth requirement of OPB is $r \times M$, and the client start-up delay is the download time of the first segment, which decreases exponentially fast as $M$ increases. The notation used for OPB and OPB/tou is summarized in Table 3.4.

Table 3.4: Notation for OPB/tou

| Symbol | Definition |
|---|---|
| $M$ | Number of segments |
| $r$ | Segment transmission rate (in units of the file playback bit rate) |
| $s$ | Maximum number of streams that clients listen to concurrently ($s \leq M$) |
| $b$ | Maximum aggregate sustainable client reception rate (in units of the file playback bit rate), $b = sr$ |
| $l_m$ | Playback duration of segment $m$ |

As with other periodic broadcast protocols, OPB transmits data on a channel even when there are no clients listening to that channel, which is inefficient in terms of server bandwidth usage, especially under light load, compared to immediate service protocols that transmit data only in response to client requests. To improve the server bandwidth efficiency of OPB, a new variant, *OPB turn off channels when unused* (OPB/tou), is proposed here. OPB/tou differs from OPB in that the server stops transmitting data on a channel whenever it detects no listeners.

Assuming Poisson request arrivals as before, the server bandwidth requirement of OPB/tou is:

$$r \sum_{m=1}^{M} \left(1 - e^{-\lambda \frac{l_m}{r}}\right), \tag{3.12}$$

where $1 - e^{-\lambda \frac{l_m}{r}}$ is the fraction of time that the channel delivering segment $i$ is not idle, which is the fraction of time that there is at least one client listening to the channel.

Figure 3.13 compares the server bandwidth requirement of OPB/tou with start-up delay $D = 0.01$ and various $s$ and $b$ values, the server bandwidth requirement of hierarchical stream merging, and the lower bound of relation (2.2) with start-up delays $D = 0.01$ and $D = 0$, as a function of the normalized file request rate $N$. As illustrated in the figure, ceasing transmission on a channel when no client is listening to it enables OPB/tou to perform comparably to immediate service on-demand streaming protocols even under light load (i.e., $N < 100$). Under heavy load, the server bandwidth usage of OPB/tou is significantly less than that of hierarchical stream merging (at the cost of non-zero client start-up delay), particularly for client receive bandwidth greater than twice the file playback bit rate (i.e. $b > 2$). Overall, OPB/tou can achieve fairly close tminimum possible server bandwidth usage.

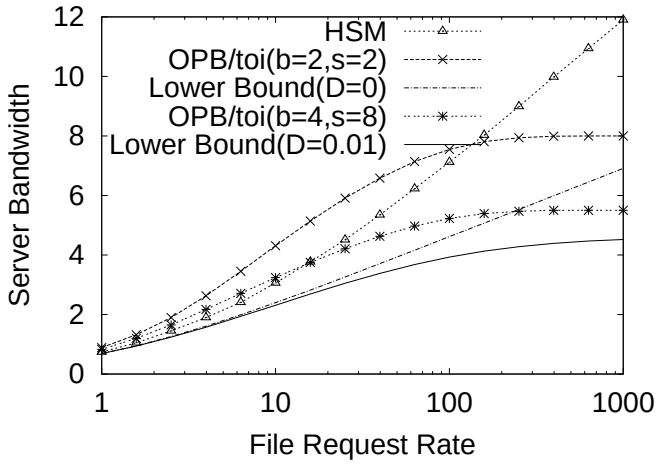



Figure 3.13: Server Bandwidth Efficiency of OPB/tou
($D = 0.01$ for OPB/tou)

### 3.5.2 Network Bandwidth of OPB/tou

The network bandwidth requirement of OPB/tou, when being used to deliver a file over a multicast delivery tree, can be computed by summing the network bandwidth requirement on each link of the tree. The required network bandwidth on a tree link is the sum of the link bandwidth required for transmitting data on each channel. Since a channel will not be delivered over a tree link whenever no clients from

the subset of clients served by the link are listening to the channel, the network bandwidth requirement on the link can be computed as:

$$r \sum_{m=1}^{M} \left(1 - e^{-\frac{l_m}{r} \sum_{i \in \mathcal{C}} \lambda_i}\right), \tag{3.13}$$

where $\mathcal{C}$ denotes the indices of the client sites served by the link.

The balanced tree topology (as shown in Figure 3.5, with equal request rate at each client site) is considered here, as previous results suggest that it yields good insight on the behaviour that could be expected in practice. The network bandwidth requirement of OPB/tou on this topology can be expressed as:

$$\frac{2^L - 1}{2^L(L-1)+1} \sum_{i=1}^{L} 2^i r \sum_{m=1}^{M} \left(1 - e^{-\frac{l_m}{r}(2^{L-i+1}-1)\frac{\lambda}{K}}\right). \tag{3.14}$$

As in Section 3.2, the required network bandwidth is expressed in units of the file playback bit rate times the average end-to-end unit bandwidth cost.

Figure 3.14 compares the network bandwidth requirement of OPB/tou with various $b$ and $s$ values, as obtained from expression (3.14), against the lower bound of relation (3.6), as a function of the start-up delay $D$. The figure suggests that OPB/tou can achieve reasonably close to the minimal network bandwidth usage.

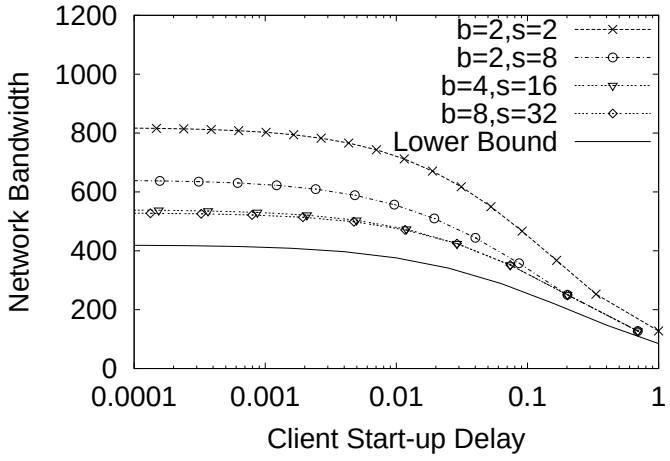


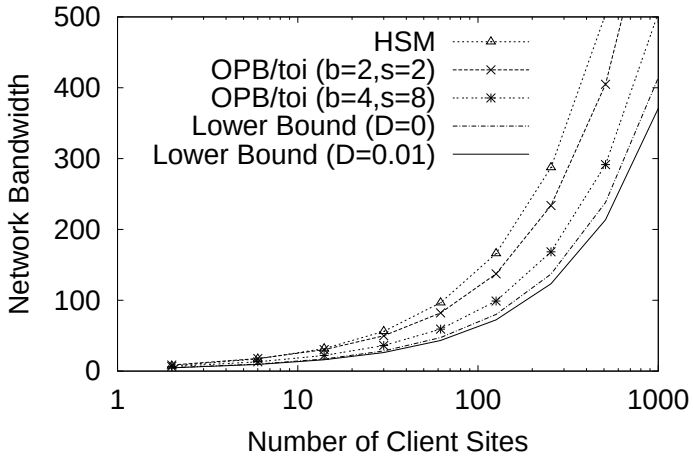

Figure 3.14: Network Bandwidth Usage of OPB/tou
(Figure 3.5 balanced binary tree topology, $N_k = 10$, $L = 9$, and $K = 1022$)

Figure 3.15 compares the network bandwidth requirement of OPB/tou with start-up delay $D = 0.01$ and various $s$ and $b$ values, to the network bandwidth requirement of network-naïve hierarchical stream merging, and the lower bound of relation (3.6) with start-up delay $D$ values of 0.01 and 0. In Figure 3.15(a), the request rate at each client site is identical and fixed, and the number of client sites $K$ is varied. In Figure 3.15(b), a fixed number of client sites is assumed and the request rate at each client site is varied. Note that in all of the scenarios considered in Figure 3.15, OPB/tou has similar or lower required network bandwidth than HSM, at the cost of non-zero client start-up delay, with particularly improved performance for client receive bandwidth of greater than twice the file playback bit rate (i.e., $b > 2$)).

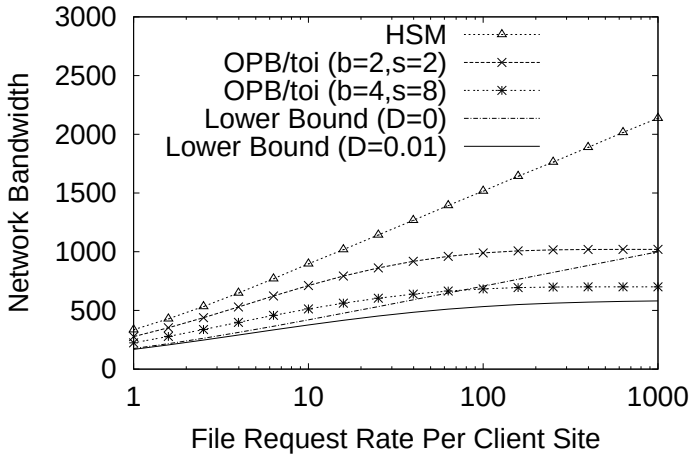
## 3.6 Summary

In this chapter, the network bandwidth requirement of scalable on-demand streaming using multicast delivery has been studied. Tight bounds on the minimum required network bandwidth for simple canonical multicast delivery trees are derived. The analysis techniques are then applied to study the network bandwidth requirements of more complex multicast delivery tree structures constructed over randomly generated large network topologies. The mathematical analysis and experimental results suggest that in practical settings the minimum required network bandwidth scales as $\Theta(K/\ln K)$ where $K$ is the number of client sites and the request rate at each client site is fixed, and for a fixed number of sites as $\Theta\left(\ln \frac{N}{ND+1}\right)$ where $N$ is the total request rate and $D$ is the start-up delay.

This chapter also explores the network bandwidth efficiency of alternative algorithms for generating multicast delivery trees, in the context of on-demand streaming. It was found that in most of the cases examined, a multicast delivery tree attempting to minimize total network bandwidth usage can only modestly (on average $3-16\%$) reduce the minimum required network bandwidth of the commonly used shortest path trees that minimize the latencies from the server to all clients.




(a) $N_k = 10$




(b) $K = 1022$ ($L = 9$)

Figure 3.15: Network Bandwidth Efficiency of OPB/tou
(Figure 3.5 balanced binary tree topology, $D = 0.01$)

This chapter also addresses the network bandwidth efficiency of practical scalable delivery protocols, specifically, the hierarchical stream merging protocol (an immediate service protocol) and a new variant of periodic broadcast protocol, OPB/tou. Simulation results show that these protocols can simultaneously achieve reasonably close to the minimum possible network and server bandwidth usage.

# Chapter 4

# Scalable On-demand Streaming of VBR Media

Audio, video, and composite media are often compressed before being stored and delivered owing to their large data volume and intensive bandwidth requirements. CBR encoding is usually applied to audio, while both CBR and VBR encoding can be used for video and composite media. CBR encoding compresses a source file at a target constant bit rate[1], but may result in varying playback quality, in particular degraded quality for scenes with extensive dynamics. VBR encoding maintains constant playback quality, but may result in substantial rate variability over time scales as long as several minutes [73]. In the context of streaming, such dramatic rate variability poses many challenges to system design (of the server and of the client set-top box) and to the resource management necessary to support jitter-free playback.

Work-ahead smoothing is a technique that addresses the problem of streaming VBR content [99, 78, 41]. Work-ahead smoothing can greatly reduce the rate variability of a VBR file by transmitting data from high rate later portions in low rate earlier portions, and buffering it in client storage until it is needed for playback. The client buffer capacity determines the extent to which a VBR file can be smoothed. Although most previous work on work-ahead smoothing assumes very constrained client buffer space (i.e., a few megabytes or less), in many multimedia streaming applications where client nodes are desktops or set-top boxes with disks, for example, the clients will have sufficient disk space to *fully* smooth a VBR file into a CBR stream (sometimes with a small start-up delay).

[1]With CBR encoding, the resulting bit rate actually fluctuates slightly around the target bit rate. A small smoothing buffer can be used to ensure a continuous transmission of the encoded file at the CBR rate [72].

Work-ahead smoothing deals with efficient transmission of a single VBR stream, while scalable delivery protocols (overviewed in Section 2.3) address how to efficiently share this stream among multiple clients that have made closely-spaced requests for the same file so as to substantially reduce the server and network bandwidth usage. Originally, scalable delivery protocols were developed for constant bit rate streaming only. Later, a number of periodic broadcast schemes, mostly based on periodic broadcast protocols for CBR media, have been proposed for VBR video [100, 86, 74, 84, 75, 56, 61, 125]. A common approach taken by these schemes is to construct a segmentation and transmission schedule such that each segment can be completely received by clients before its playback begins, in which case it can be fully smoothed and delivered as a CBR stream [86, 84, 56, 61, 125]. However, in one previous study, it was found that using work-ahead smoothing in periodic broadcast protocols for VBR media streaming may incur more server bandwidth usage than delivering the media unsmoothed [56]. To date, no research has investigated scalable streaming of VBR files using immediate service protocols.

A straightforward approach would be to fully smooth a VBR file and then directly apply an existing immediate service protocol to the smoothed CBR stream. However, as will be shown in this chapter, there is a fundamental conflict between work-ahead smoothing and scalable on-demand streaming, particularly with immediate service protocols. Scalable on-demand streaming protocols require frequent transmissions of the earlier portions of a media file, so as to provide low (or zero) start-up delay. On the other hand, these protocols achieve bandwidth efficiency by relatively infrequently delivering the later portions of the file and sharing these transmissions among many clients. Work-ahead smoothing involves moving data from later high rate portions of a media file to earlier low rate portions, and therefore has the effect of increasing the amount of data that needs to be frequently delivered and decreasing the amount of data that is infrequently delivered.

The work presented in this chapter aims at understanding the impact of this conflict on the design of immediate service scalable streaming protocols for VBR media. This study focuses on server bandwidth efficiency. Network bandwidth efficiency can

be analyzed using the techniques developed in Section 3.2. The conclusions drawn from server bandwidth analyses are qualitatively applicable to network bandwidth. Therefore, network bandwidth analysis is omitted from this chapter.

The remainder of this chapter is organized as follows. Section 4.1 provides background on work-ahead smoothing. Section 4.2 describes the characteristics of a variety of variable bit rate profiles to be used in the following analyses and experiments, and evaluates their start-up delays and storage space requirements for full smoothing. Section 4.3 studies the impact of work-ahead smoothing on the server bandwidth requirement of scalable on-demand streaming. Section 4.4 proposes a new immediate service scalable streaming protocol for VBR media, *VBR bandwidth skimming* (VBRBS), and a variant of this protocol that supports delivery to clients with limited storage capacity. Section 4.5 evaluates the performance of VBRBS against that of the straightforward approach that applies an existing immediate service protocol to a smoothed stream. Section 4.6 summarizes the work presented in this chapter.

## 4.1 Work-ahead Smoothing

Work-ahead smoothing, *statistical multiplexing*, and *temporal buffering* are techniques for reducing the rate variability of VBR media transmissions. Statistical multiplexing aggregates a number of VBR streams on a single transmission channel with the expectation that the rate variability of the aggregate stream will be less than that of the component streams. In particular, it is expected that the peaks in rate of the component streams will not occur at the same times and therefore the allocated channel capacity can be significantly less than the sum of the peak rates. If the bandwidth of the aggregate stream exceeds the channel capacity, packets have to be dropped. To keep the packet loss ratio low, an admission control mechanism can be employed to restrict the initiation of new streams on the channel.

Temporal buffering introduces buffers along the path from server to client. These buffers can absorb bursts of traffic from VBR streams, but will also unavoidably introduce delay. Moreover, packet loss can still happen if a buffer overflows.

Work-ahead smoothing transmits data from high rate periods ahead of its playback time. It can be implemented by converting a VBR transmission into a piecewise-CBR transmission. Each CBR piece is defined by a bandwidth requirement and a duration. Various work-ahead smoothing algorithms differ in how the bandwidth and the duration of each CBR piece are determined [42]. However, they are all subject to two constraints: (1) a client should always have sufficient buffer space to receive the prefetched data (i.e., the client buffer cannot be allowed to overflow); and (2) a client should always have data in the buffer to be retrieved (i.e., the client buffer cannot be allowed to underflow). Since buffer overflow is prevented, work-ahead smoothing does not cause packet loss. Compared to temporal buffering, work-ahead smoothing may only introduce a short start-up delay to smooth the initial portion (e.g., a few frames) of a video file if its required bit rate is higher than the target smoothing rate.

Although a number of work-ahead smoothing algorithms have been developed [42], three basic types of smoothing are of interest in this study: *full smoothing* creates a constant bit rate stream at some specified rate; *minimal smoothing* requires the minimal amount of client buffer space when smoothing a file so as not to exceed a given peak rate; and *optimal smoothing* achieves the lowest peak bit rate and the smallest bit rate variability given a fixed amount of client buffer space.

### Full Smoothing

If the client buffer is sufficiently large, a VBR file can be fully smoothed and delivered as a CBR stream. Depending on the rate of the CBR stream, and the rate profile of the media file, a start-up delay may be required so as to ensure that each media unit is received by its playout point.

### Minimal Smoothing

Minimal smoothing smoothes a VBR file just enough to bring its peak rate under a given rate constraint [42, 107].

Minimal smoothing can be accomplished with a single pass through the VBR file, from the end to the beginning. For each media unit examined, if its rate is greater than the rate constraint, the rates of the preceding units leading up to the current unit are increased (i.e., data from the media unit will be prefetched) so as to reduce the rate of the current unit to the constrained rate.

Minimal smoothing uses the minimum possible client buffer space to achieve a target peak rate. If the client buffer space is insufficient to support minimal smoothing, either the constrained rate or the client buffer capacity must be increased.

### Optimal Smoothing

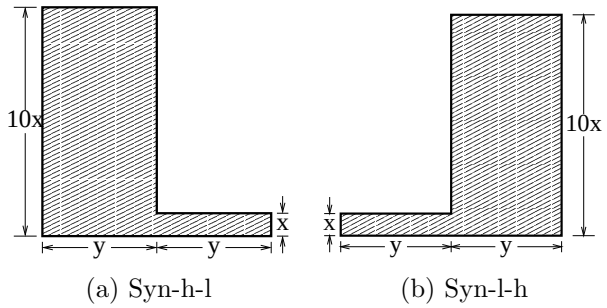
Optimal smoothing smoothes a VBR file into a sequence of CBR segments such that the peak rate and the rate variability are the minimum achievable for a given client buffer size [99].

Starting from the beginning of a VBR file, the optimal smoothing algorithm establishes a longest possible CBR segment. If this segment must end because the client buffer is about to overflow, a new CBR segment with a decreased rate will start at the latest time when the client buffer is empty. On the other hand, if the segment must end due to client buffer underflow, a new CBR segment with an increased rate will start at the latest time when the client buffer is full. This strategy minimizes the change in rate from that of the current CBR segment, thereby resulting in the smoothest sequence of CBR segments among all feasible transmission schedules.

## 4.2 VBR Trace Characteristics

This study uses the bit rate profiles of two synthetic objects, two composite objects (consisting of various media types [107]), sixteen 27-minute MPEG-1 encoded short video clips [98], and a 2-hour full MPEG-1 compressed video *star wars* [50].

The two synthetic objects are composed of an initial part and a latter part with equal lengths. For the first synthetic object (*Syn-h-l*), the bandwidth of the initial part is ten times greater than that of the latter part. For the second synthetic object (*Syn-l-h*), the bandwidths of the two parts are reversed. Figure 4.1 shows the bit rate profiles of these two synthetic objects.

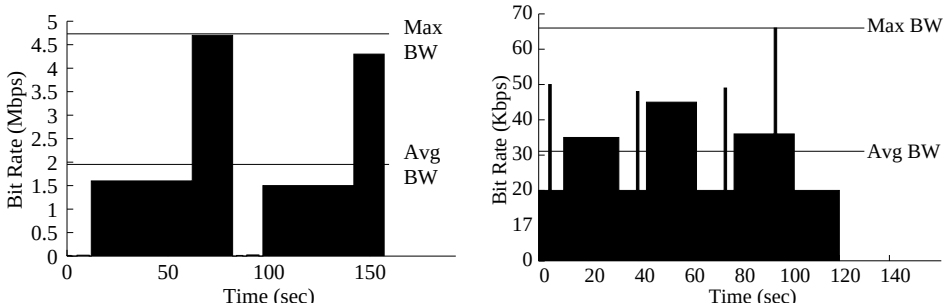



(a) Syn-h-l (b) Syn-l-h

Figure 4.1: Synthetic Object Bit Rate Profiles

The two composite bit rate profiles are obtained from [107]. *Composite 1* is a report of a swimming competition, including not only the global views but also the close-ups of the leaders. *Composite 2* is a combination of video, images and narrations showing tourism sites around Washington D.C.. Figure 4.2 depicts their bit rate profiles.

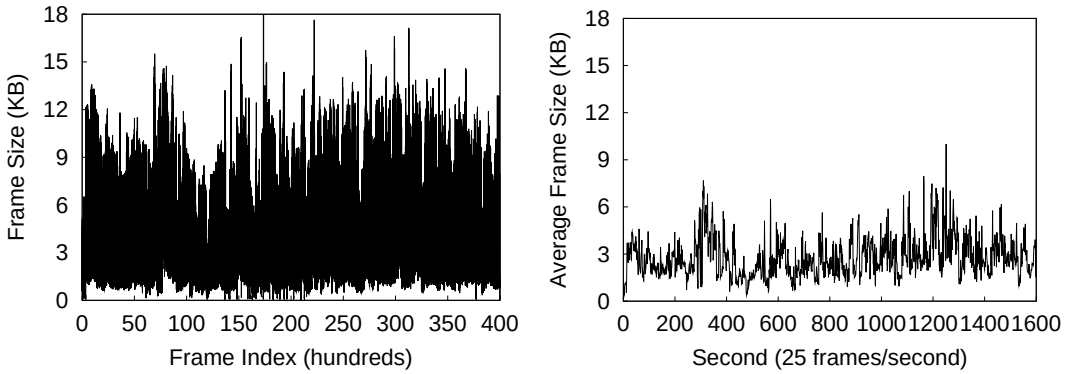
The 16 short video clips are created using the UC Berkeley MPEG-1 software encoder. Each video segment contains 40,000 frames, representing 26.7 minutes of video at 25 frames/second. The encoder input is $384 \times 288$ pixels with 12 bit color information. The 121 minute *star wars* video consists of 171,000 frames with a frame rate of 24 frames/second (i.e., the original film rate). The original video is captured as $408 \times 508$ pixels, and then interpolated and filtered to standard CIF frame size, which is $240 \times 352$ (Luminance - Y) and $120 \times 176$ (Crominance - U & V). All video files use the sequence of MPEG I, P and B frames IBBPBBPBBPBB. Figure 4.3



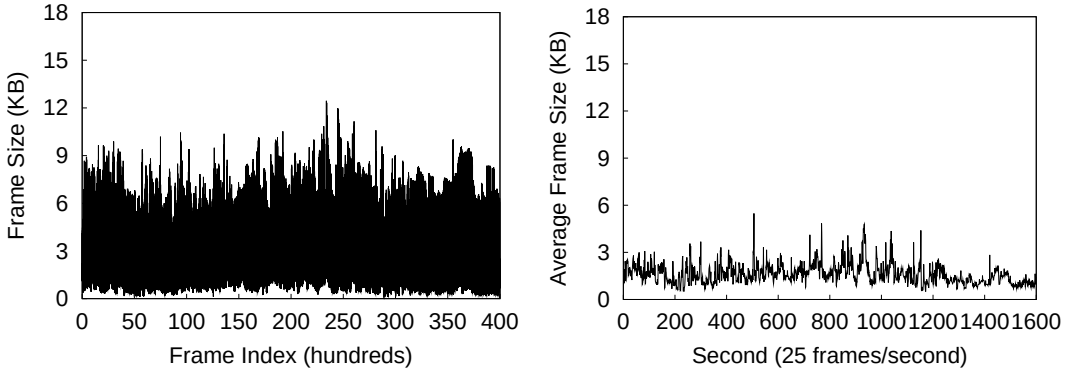

(a) Composite 1 (b) Composite 2

Figure 4.2: Composite Object Bit Rate Profiles

shows the bit rate profiles of the two short video clips Asterix and Dinosaur. The left-side graphs plot the frame size, and the right-side graphs plot the averaged frame size over a second interval. These graphs suggest that the video clips have considerable bit rate variability.




(a) Asterix




(b) Dinosaur

Figure 4.3: Video Clip Bit Rate Profiles

Table 4.1 summarizes the characteristics of all of the bit rate profiles used in this study. The table also lists the required start-up delay (as a percentage of the playback duration) and buffer space (as a percentage of the file size) when fully smoothing a VBR bit rate profile to a CBR stream at a rate equal to the file's average bit rate ("*Avg*") and 1:2 times the average bit rate ("*1.2Avg*"), respectively. The factor 1.2 is chosen because most of the VBR files do not require any start-up delay when fully smoothed to that rate. The last column of Table 4.1 shows the minimum smoothing rate that can reduce the start-up delay to zero, relative to the file's average bit rate.

Table 4.1: Object Characteristics

| Video Name | Avg Rate (Mbps) | Frame Sizes (KBytes) | | | | Start-up Delay (%) | | Storage (%) | | Rate/Avg (Delay=0) |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Avg | Max | Min | Std | Avg | 1.2Avg | Avg | 1.2Avg | |
| asterix | 0.56 | 2.79 | 18.4 | 0.04 | 2.52 | 0.43 | 0.00 | 6.23 | 19.2 | 1.02 |
| dinosaur | 0.33 | 1.63 | 15.0 | 0.11 | 1.84 | 6.62 | 0.04 | 7.26 | 12.4 | 1.21 |
| fuss | 0.68 | 3.39 | 23.4 | 0.31 | 3.25 | 0.91 | 0.00 | 2.74 | 16.7 | 1.07 |
| lambs | 0.18 | 0.91 | 16.8 | 0.04 | 1.40 | 4.09 | 0.66 | 10.0 | 16.7 | 3.83 |
| movie2 | 0.36 | 1.79 | 21.6 | 0.03 | 2.36 | 3.85 | 0.20 | 5.26 | 17.5 | 1.26 |
| mrbean | 0.44 | 2.21 | 28.6 | 0.04 | 2.58 | 3.45 | 0.74 | 9.45 | 21.3 | 2.73 |
| mtv1 | 0.62 | 3.08 | 28.7 | 0.05 | 2.88 | 4.87 | 0.03 | 6.34 | 13.5 | 1.81 |
| mtv2 | 0.49 | 2.47 | 31.4 | 0.06 | 2.68 | 6.31 | 0.02 | 11.8 | 21.2 | 1.29 |
| news2 | 0.38 | 1.92 | 23.7 | 0.03 | 2.44 | 1.96 | 0.01 | 4.68 | 16.4 | 1.42 |
| race | 0.77 | 3.84 | 25.3 | 0.52 | 2.65 | 1.83 | 0.07 | 2.90 | 17.1 | 1.38 |
| simpsons | 0.46 | 2.32 | 30.1 | 0.04 | 2.58 | 3.98 | 1.50 | 3.98 | 15.6 | 1.90 |
| soccer | 0.63 | 3.14 | 23.8 | 0.37 | 2.66 | 1.33 | 0.16 | 4.85 | 16.9 | 1.93 |
| superbowl | 0.59 | 2.94 | 17.6 | 0.04 | 2.34 | 1.38 | 0.00 | 4.27 | 15.8 | 1.06 |
| talkshow1 | 0.36 | 1.82 | 13.4 | 0.26 | 2.06 | 2.44 | 1.16 | 4.02 | 18.3 | 2.58 |
| talkshow2 | 0.45 | 2.24 | 16.6 | 0.45 | 2.28 | 1.99 | 0.01 | 4.27 | 18.5 | 1.52 |
| terminator | 0.27 | 1.36 | 9.95 | 0.04 | 1.27 | 2.02 | 0.25 | 2.97 | 15.5 | 1.30 |
| starwars | 0.37 | 1.95 | 23.2 | 0.06 | 2.27 | 0.51 | 0.00 | 6.58 | 21.1 | 1.18 |

| Name | Avg Rate (Mbps) | Peak Rate (Mbps) | Start-up Delay (%) | | Storage (%) | | Rate/Avg (Delay=0) |
|---|---|---|---|---|---|---|---|
| | | | Avg | 1.2Avg | Avg | 1.2Avg | |
| composite 1 | 1.95 | 4.70 | 4.62 | 0.00 | 18.0 | 26.5 | 1.09 |
| composite 2 | 0.03 | 0.07 | 5.35 | 0.00 | 7.73 | 12.2 | 1.10 |
| synthetic 1 | 0.55 | 1.0 | 40.9 | 25.8 | 40.9 | 30.9 | 1.82 |
| synthetic 2 | 0.55 | 1.0 | 0.00 | 0.00 | 40.9 | 50.9 | 1.00 |

## 4.3 Impact of VBR on Server Bandwidth

### 4.3.1 Server Bandwidth for Streaming VBR Media

As described in Section 2.4, a tight lower bound on the server bandwidth requirement for on-demand streaming of a CBR file is given by:

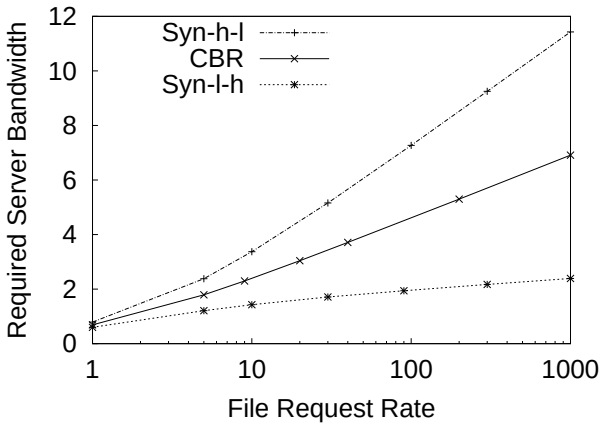
$$B_{min}^{srv} = \int_0^T \frac{1}{x+d+1/\lambda} dx = \ln\left(\frac{T}{d+1/\lambda}+1\right) = \ln\left(\frac{T\lambda}{d\lambda+1}+1\right), \qquad (4.1)$$

where $B_{min}^{srv}$ is in units of the file playback bit rate, $d$ is the maximum client start-up delay, $T$ is the file playback duration, and $\lambda$ is the file request rate.

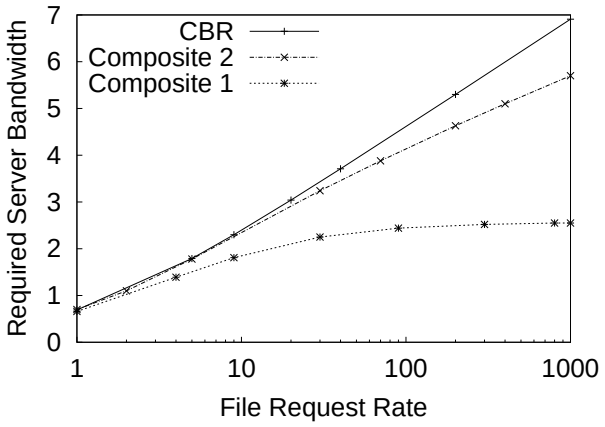
A lower bound on the server bandwidth requirement for on-demand streaming of a VBR file can be derived from the above equation as follows. Defining the VBR file as a sequence of $W$ CBR segments, $<< 0, b_0 >, < t_1, b_1 >, \cdots, < t_{w-1}, b_{w-1} >>$, where segment $i$ starts at time $t_i$ ($0 < t_i < T$ for $i > 0$) with bandwidth requirement $b_i$ ($b_i \geq 0$; $b_i \neq b_{i+1}$), the lower bound (in units of the file's average bit rate $b$) can be computed as:

$$\begin{aligned} B_{min}^{VBR} &= \frac{b_0}{b}\int_0^{t_1} \frac{1}{x+d+1/\lambda} dx + \frac{b_1}{b}\int_{t_1}^{t_2} \frac{1}{x+d+1/\lambda} dx + \cdots + \frac{b_{w-1}}{b}\int_{t_{w-1}}^{T} \frac{1}{x+d+1/\lambda} dx \\ &= \frac{b_0}{b}\ln\frac{(t_1+d)\lambda+1}{d\lambda+1} + \frac{b_1}{b}\ln\frac{(t_2+d)\lambda+1}{(t_1+d)\lambda+1} + \cdots + \frac{b_{w-1}}{b}\ln\frac{(T+d)\lambda+1}{(t_{w-1}+d)\lambda+1}. \end{aligned} \qquad (4.2)$$

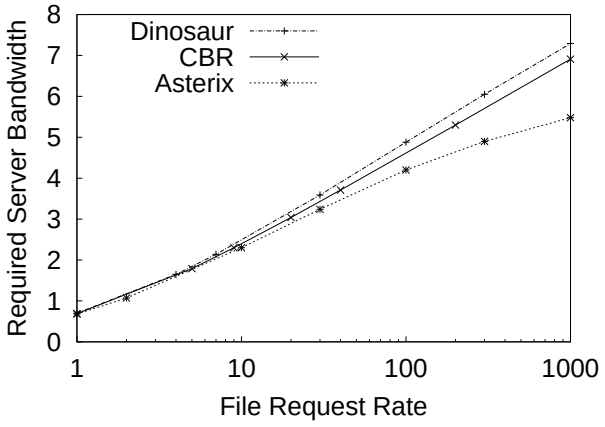
Figure 4.4 compares the lower bound of equation (4.2) for various VBR bit rate profiles, to the lower bound of equation (4.1) for CBR media, for immediate service ($d = 0$). In Figure 4.4(a), the bandwidth required for the synthetic object Synh- l is greater than that for a CBR file, which in turn is more than that for the synthetic object Syn-l-h. As illustrated in this figure, the required server bandwidth for delivering a VBR file is largely determined by the bit rates of the initial portions of the file, especially under high client request rate (i.e., for hot files), since these portions are transmitted much more frequently than the later portions. This result also explains why Composite 2 and Dinosaur demand more server bandwidth than Composite 1 and Asterix in Figures 4.4(b) and 4.4(c), respectively.




(a) Synthetic Objects




(b) Composite Objects




(c) Short Video Clips

Figure 4.4: Server Bandwidth for Unsmoothed VBR Files ($d = 0$)

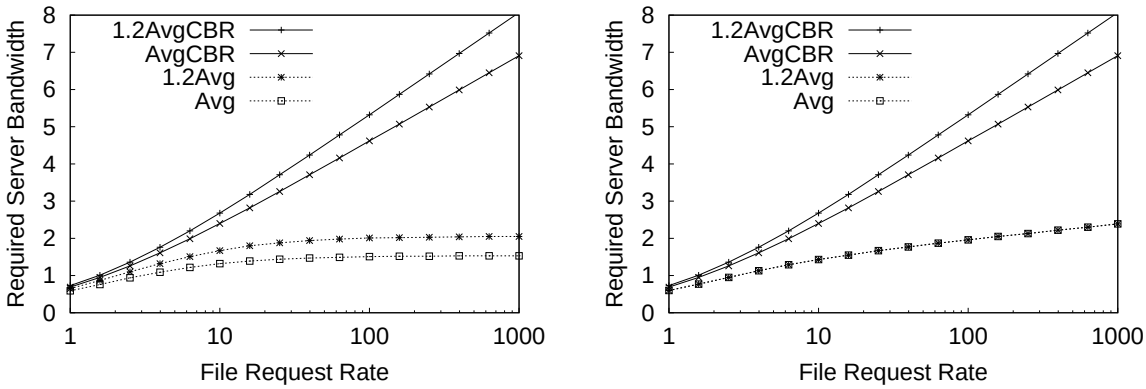
### 4.3.2 Impact of Work-ahead Smoothing on VBR Delivery

A straightforward approach to scalable delivery of a VBR media file is to smooth it into a CBR stream and then apply an existing scalable delivery technique to it. Previous research has found that work-ahead smoothing may result in inefficient server bandwidth usage when used with periodic broadcast protocols to deliver VBR media [56]. This section employs the lower bound of equation (4.2) to analyze the impact of work-ahead smoothing on the server bandwidth usage for streaming VBR files with immediate service protocols.

Figure 4.5 compares the minimum required server bandwidth for streaming a fully smoothed VBR media file (from equation (4.1)), to that for streaming the VBR file unsmoothed (from equation (4.2)), for the two synthetic objects. Note from Table 4.1 that delivery of syn-h-l, when fully smoothed to its average bit rate or 1.2 times this bit rate, requires a start-up delay. This is not, however, a start-up delay that can be exploited by a scalable delivery protocol, since it is a delay for when the media stream starts rendering at the client rather than a server-side delay. Thus, when applying equation (4.1), $d = 0$ is used. To make a fair comparison with scalable delivery of the unsmoothed VBR file, however, the same start-up delay should be allowed, and in this case, owing to the ability of the server to deliver an initial high rate portion of a file at that high rate, the delay can be a server-side delay (i.e., $d > 0$).

Figure 4.5 presents results for when a VBR file is fully smoothed to the file's average bit rate ("*AvgCBR*"), unsmoothed but with the same start-up delay for syn-h-l as required when smoothed to the average bit rate ("*Avg*"), fully smoothed to 1.2 times the file's average bit rate ("*1.2AvgCBR*"), and unsmoothed but with the same start-up delay for syn-h-l as required when smoothed to 1.2 times the average bit rate ("*1.2Avg*"). The figure illustrates how delivering a fully smoothed VBR file can require substantially more bandwidth than delivering the file as is (with the same start-up delay, if any, added). As expected, increasing the target rate for full smoothing, for example from the file's average bit rate to 1.2 times

that rate (implying reduced start-up delay for syn-h-l), increases the required server bandwidth with fully-smoothed delivery, since smoothing to a higher target rate pushes more data to the beginning of the file. The bandwidth is also increased for unsmoothed delivery when a lower start-up delay is used for the comparison (as in the case of syn-h-l).



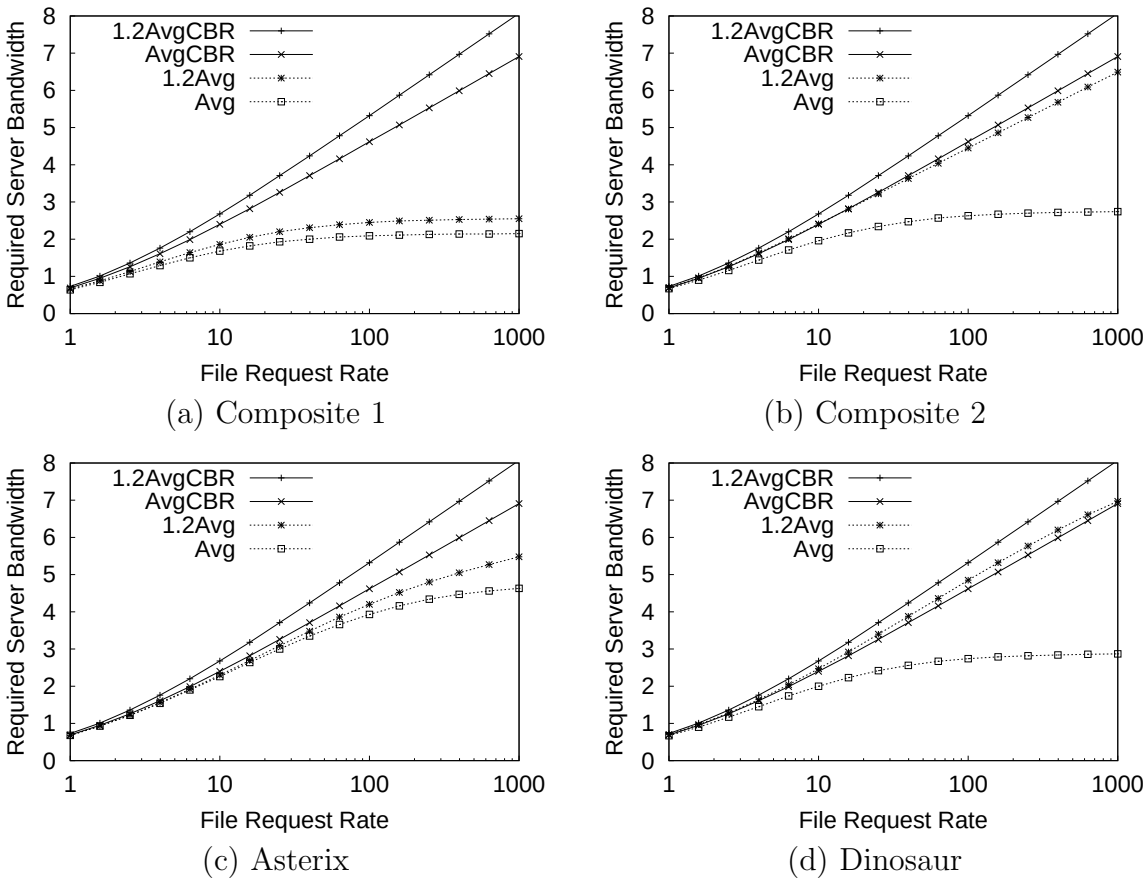

(a) Syn-h-l (b) Syn-l-h

Figure 4.5: Server Bandwidth for Smoothed Synthetic Bit Rate Profiles ($d = 0$ except for delivery of an unsmoothed VBR file for which the delay is non-zero with work-ahead smoothing)

Figure 4.6 presents the corresponding results for the two composite objects and the two short video clips (Asterix and Dinosaur). Similar conclusions can be drawn from these results – that is, work-ahead smoothing conicts with on-demand streaming of VBR media in that smoothing increases the server bandwidth usage.

## 4.4 VBR Bandwidth Skimming

The previous section has analyzed the minimum required server bandwidth for on-demand streaming of VBR media, and has found that it is potentially much more efficient to deliver a VBR file unsmoothed, rather than first fully smoothing the file. This result is consistent with and supplementary to the previous finding that work-ahead smoothing is inefficient with respect to the resulting bandwidth usage when used with periodic broadcast protocols to deliver VBR files [56]. On the other




(a) Composite 1 (b) Composite 2

(c) Asterix (d) Dinosaur

Figure 4.6: Server Bandwidth for Smoothed Composite and Video Files ($d = 0$ except for delivery of an unsmoothed VBR file for which the delay is non-zero with work-ahead smoothing)

hand, a CBR stream is much easier to transmit and manage than a VBR stream. In this section, a new immediate service protocol for VBR media, VBR bandwidth skimming (VBRBS), is proposed. VBRBS transmits a CBR stream over the network and at the same time exploits the bit rate profile of a VBR file to achieve server bandwidth efficiency.

### 4.4.1 Protocol Description

VBR bandwidth skimming is based on the *latest patch* bandwidth skimming protocol (described in Section 2.3.3). As illustrated in Figure 4.7, VBR bandwidth skimming differs from latest patch in the following three aspects.

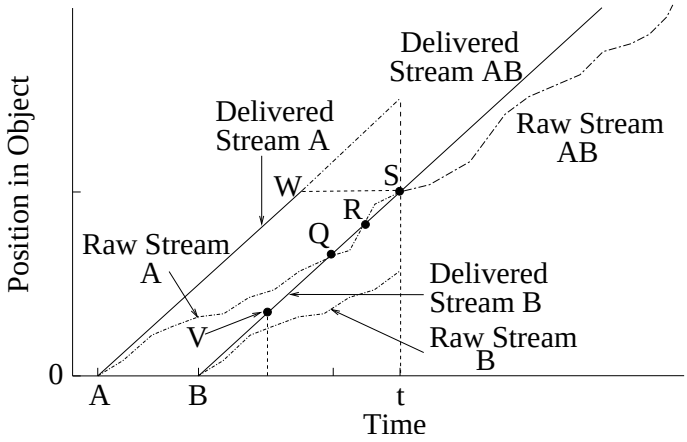



Figure 4.7: VBR Bandwidth Skimming

First, in VBRBS, the *raw* stream (shown by a dashed line) that represents playout progress is no longer a CBR stream as in latest patch, but is a VBR stream. Depending on the rate of the delivered stream and of the initial portion of the raw stream, a start-up delay may be required. In the following description of VBRBS, any required start-up delay is (conceptually) treated by the addition of media elements (e.g., frames) of zero size at the beginning of the file, the "playout" of which constitutes the start-up delay.

Second, in latest patch, a *merge* happens when a later client's delivered stream reaches the same position in the file, defined as the *merge point*, as that reached by the previous client's raw stream. According to this definition, points Q, R, and S in Figure 4.7 are all potential merge points. However, between points R and S, client A's raw stream is at later positions in the file than client B's delivered stream. Therefore, if client A and client B merge before point S (such as at point Q or R), the merged group's delivered stream, which would be client B's delivered stream, would not provide the data between R and S in time for client A's playout. For this reason, points Q and R cannot be merge points. Point S, however, can be the merge point because after point S, the merged group's delivered stream would always be at a later position in the file than the merged group's raw stream. In VBRBS, the merge point is defined as the *earliest* point in the file at which the later client's delivered stream and the previous client's raw stream intersect, and after which the

merged group's delivered stream is always at the same or a later position in the file than its raw stream.

Third, in Figure 4.7, after clients A and B merge, any data received by client A (via its delivered stream) beyond point W, which is at the same position in the file as the merge point S, will be delivered to the merged group again. Figure 4.8 depicts the server bandwidth used to transmit this *redundant* data, assuming that delivered streams are terminated only at the end of the file or at merge points (for client A in Figure 4.7 at time $t$), as a percentage of the total server bandwidth required for streaming the two composite objects and the two short video clips Asterix and Dinosaur. As can be seen from the figure, for hot files (i.e., file request rate exceeding 100), 10–18% of the total server bandwidth is redundant. This observation motivates a strategy used in VBRBS named *early commit*. Using early commit, once a later client's (or group's) delivered stream goes beyond a pre-defined *commit point*, for example the point V in Figure 4.7 which corresponds to 1/3 of the duration from the start of client B's delivered stream to the merge time $t$, the later client will commit this merge. Once a merge is committed, the previous client's (or group's) delivered stream (i.e., in Figure 4.7 the client A's delivered stream) will be terminated when it reaches the point (point W in Figure 4.7) that is at the same position in the file as the merge point, implying that the data beyond this point is delivered only once by the server. A potential disadvantage of early commit is its impact on the merging structure. With a merging policy such as ERMT or CT, new request arrivals can lead to substantial changes in the ordering of merges, but with early commit, the merge structure becomes somewhat more static. The performance benefits of early commit have been found to outweigh any performance impact from the more static merging structure, however, as seen in Section 4.4.3.

### 4.4.2 Determining Merge Points

A straightforward approach to determining a merge point would be to go through the VBR bit rate profile from the beginning to the end (e.g., frame by frame), and

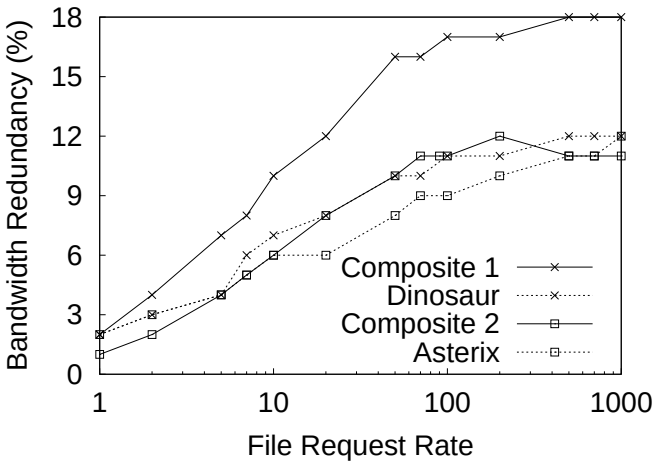



Figure 4.8: Server Bandwidth Used for Transmitting Redundant Data

find the *earliest* point after which the delivered stream of the later client or group of clients is always at the same or later file position as the raw stream of the previous client or group of clients. For a VBR file of $F$ frames, the time complexity of this approach is $O(F)$, which may be too costly to allow this approach to be used in real time.

Merge points can be determined more efficiently by pre-computing a merge table for each VBR file to be delivered. This table contains two fields: a separation time $t$ and a merge frame number $i$ (or, more generally, media item number $i$), indicating that if the time separating client request arrivals is $t$, the two clients can merge at frame $i$. Once the merge table is created, determining the merge point for any two clients whose requests were made at times $t_1$ and $t_2$ respectively (or two groups of clients whose earliest/latest client requests were made at times $t_1/t_2$) is as simple as searching the table for the frame number i that corresponds to the *smallest* separation time $t$ that is greater than or equal to $t_2 - t_1$. The merge time, then, can be computed as $t_2 + i/R$, where $R$ is the frame rate. Although the granularity of merge points employed in this study is frames, it is quite straightforward to generate merge tables at coarser granularities, for example at the GOP level. There is a trade-off between the accuracy of the calculation of the earliest possible merge time and the space requirement of the merge table.

Figure 4.9 outlines an algorithm for generating a merge table for a VBR file. Figure 4.9(a) lists the notation used by the algorithm, and Figure 4.9(b) presents pseudocode for the algorithm itself.

| | |
|---|---|
| $r$ | : delivered stream rate |
| $F$ | : number of frames |
| $b_i$ | : bit rate of frame $i$ |
| $R$ | : frame rate |
| $S_i$ | : total # of bits in first $i$ frames |
| $F_i$ | : frame $i$ |

(a) Notation

```
Procedure Generate_merge_table(r)
1.   i ← F
2.   while (i > 0) {
3.      while (b_i < r) {
4.         add [i/R − S_i/d, i] to table
5.         i ← i − 1 }
6.      j ← i − 1
7.      while((slope(F_j, F_i) > r)
                    AND(j > 0))
8.         j ← j − 1
9.      i ← j }
```

(b) Algorithm Outline

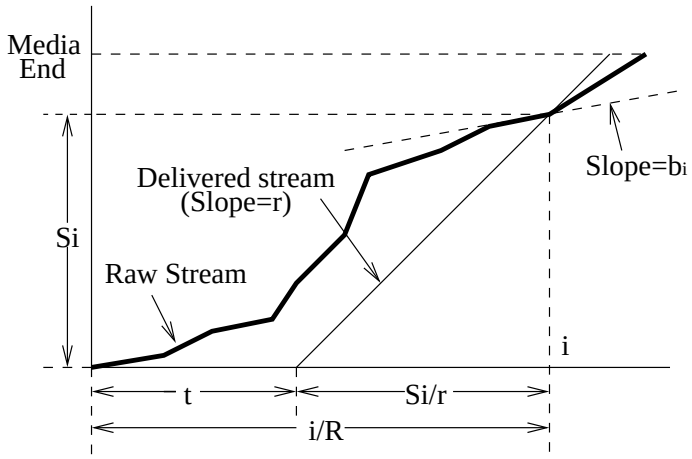
Figure 4.9: Algorithm for Generating a Merge Table

This algorithm goes through the VBR bit rate profile from the end to the beginning, frame by frame, and compares the bit rate of the current frame $i$ (i.e., $b_i$) against the delivered stream rate $r$:

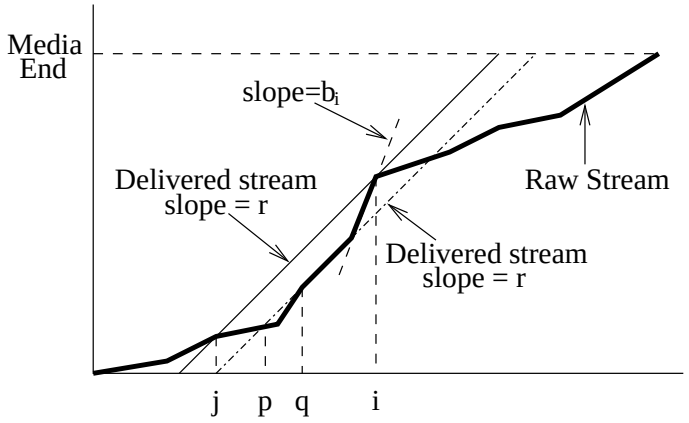
- Case 1: $b_i \leq r$ (Lines 3–5). As illustrated in Figure 4.10(a), if a delivered stream and a raw stream are both at frame i at the same point in time, then the delivered stream will be at the same or a later position in the file than the raw stream, for the remainder of the file. Therefore, frame i is a valid merge point for any two clients whose request times are separated by at most $i/R - S_i/r$, where $i/R$ is the time duration from the beginning of the previous client's raw stream until the merge time, and $S_i/r$ is the time duration from the start of the later client's delivered stream until the merge time.

- Case 2: $b_i > r$ (Lines 6–9). This case is illustrated in Figure 4.10(b). Lines 7-8 find the next frame $j$ such that if a delivered stream and a raw stream are both at frame $j$ at the same point in time, then the delivered stream will be at the same or a later position in the file than the raw stream, for the remainder

of the file. Any frame between frame $i$ (including $i$) and frame $j$ cannot be a VBRBS merge point, because any clients that could merge at such a frame could also merge at the earlier frame $j$. Therefore, the next possible merge point is frame $j$.




(a) Bit Rate of Frame $i \leq$ Delivered Stream Rate $r$




(b) Bit Rate of Frame $i >$ Delivered Stream Rate $r$

Figure 4.10: Two Scenarios in the Algorithm of Figure 4.9

### 4.4.3 Determining Commit Points

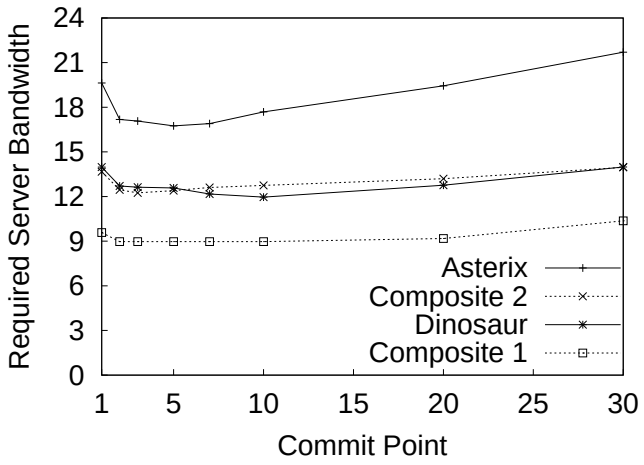
Section 4.4.1 introduced the *early commit* policy to reduce the redundant serve bandwidth used to transmit data to a previous client beyond the position in the file corresponding to that of the merge point. When early commit is employed, it

is necessary to determine a proper commit point. The closer a commit point is to the beginning of the later client's (or group's) delivered stream, the less chance there is that the previous client's (or group's) delivered stream will transmit data beyond the position corresponding to the merge point. However, a closer commit point also implies a more static merging structure, which may have an associated server bandwidth cost.

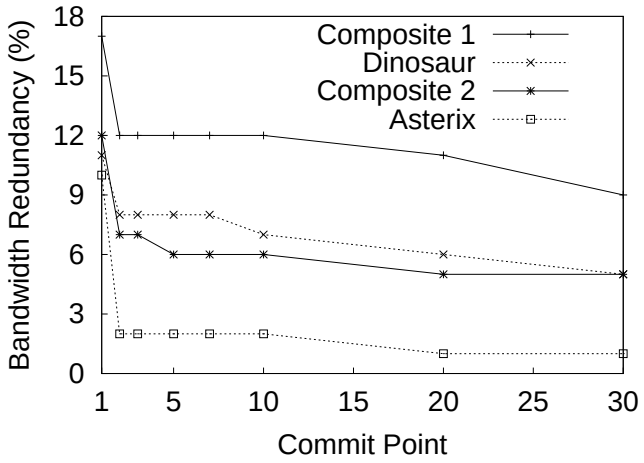
Figure 4.11(a) depicts the required server bandwidth using VBRBS for different commit point placements, for the two composite objects and the two short video clips Asterix and Dinosaur. The commit point placement is expressed as an integer $n$ where $1/n$ is the ratio of the duration from the beginning of the later client's (or group's) delivered stream to the commit time, to the duration from the beginning to the merge time. (In this definition, the "beginning" of the delivered stream for a group of clients is considered to be the merge time at which the group was formed.) In the figure, the client storage space is assumed to be unlimited, the client receive bandwidth ($crb$) is equal to 1.2 times the file's average bit rate, and the client request rate $N$ is set to 200. Note that a commit point placement of 1 indicates that no early commit is employed. A commit point placement equal to 2 means that once the later client is half way towards merging with a previous client, it will commit the merge. As can be seen from the figure, an early commit policy can reduce the server bandwidth usage by as much as 15%. Moving the commit point earlier than half way has little further impact on the required server bandwidth, up to a commit point placement of 10. Moving the commit point even earlier can increase the required server bandwidth, owing to the server bandwidth cost of a more static merging structure.

Figure 4.11(b) depicts the bandwidth used for redundant data with various commit point placements. The figure confirms the observations from Figure 4.11(a) that using early commit can significantly reduce the server bandwidth used for transmitting redundant data, and a commit point placement greater than 2 (i.e., half way to the merge) does not considerably further reduce this bandwidth usage. Overall, Figure 4.11 suggests that a commit point placement less than 10 should be used.

In the following experiments, the commit point placement of 3 is chosen, implying that once a later client is a third of the way towards a merge with a previous client, it will commit the merge.




(a) Server Bandwidth Requirement




(b) Bandwidth Used for Redundant Data

Figure 4.11: Server Bandwidth Usage for Varying Commit Point Placements ($crb = 1.2$, $N = 200$)

### 4.4.4 Accommodating Limited Client Storage

The VBRBS protocol as described so far assumes unlimited client buffering capacity (more specifically, buffer space equal to $\max_i \left(S_i - S_{j(i)}\right)$ where $j(i) = \left\lceil \frac{FS_i}{(crb)S_F} \right\rceil$ where $F$ denotes the total number of frames or media units and Si denotes the total number of bits in the first $i$ frames.), which may require use of disk buffering.

Although this requirement can be met by desktop PCs, laptops, and set-top boxes at low cost, it may not be feasible for some current mobile and handheld devices, such as Palm Pilots and Pocket PCs, which have only a limited amount of memory (e.g. 8-256 Mbytes).

VBRBS can be easily extended to deliver VBR files to clients with limited storage capacity. As with the client receive bandwidth, the following discussion assumes homogeneous client buffer capacities, although extension to heterogeneous clients is possible. One approach is as follows. Whenever the client (or all of the clients in a group) has available buffer space, the rate of the delivered stream is equal to client receive bandwidth. When the buffer fills (for a group of clients, this will be the buffer of the client whose request time was the latest among the clients in the group), the rate of the delivered stream is reduced to the minimum of the client receive bandwidth and the playback bit rate. This strategy maximizes the chance that a later client can catch up to a previous client (because at each point in time the rate of the delivered stream is as high as possible), but may result in a delivered stream with high bit rate variability. This approach is termed simply VBRBS in the following.

In order to reduce the rate variability of delivered streams, *VBRBS with smoothing* applies optimal smoothing [99] to these streams. The optimal smoothing algorithm first computes an upper bound stream and a lower bound stream based on the rate profile of the delivered stream and the client buffer capacity, and then finds a sequence of CBR segments within the bounds that has both the minimum peak rate and the minimum rate variability. In *VBRBS with smoothing*, the smoothed stream used in the variant described above is considered as the upper bound stream, and the minimally smoothed stream (at the client receive bandwidth) is considered as the lower bound stream, but with one modification, that is, the duration of the minimally smoothed stream is modified if necessary to be equal to that of the fully smoothed stream. This truncation is done by supposing that all data consumed after the time corresponding to the end of the fully smoothed stream is consumed instantaneously at that time.

## 4.5 Performance Evaluation of VBRBS

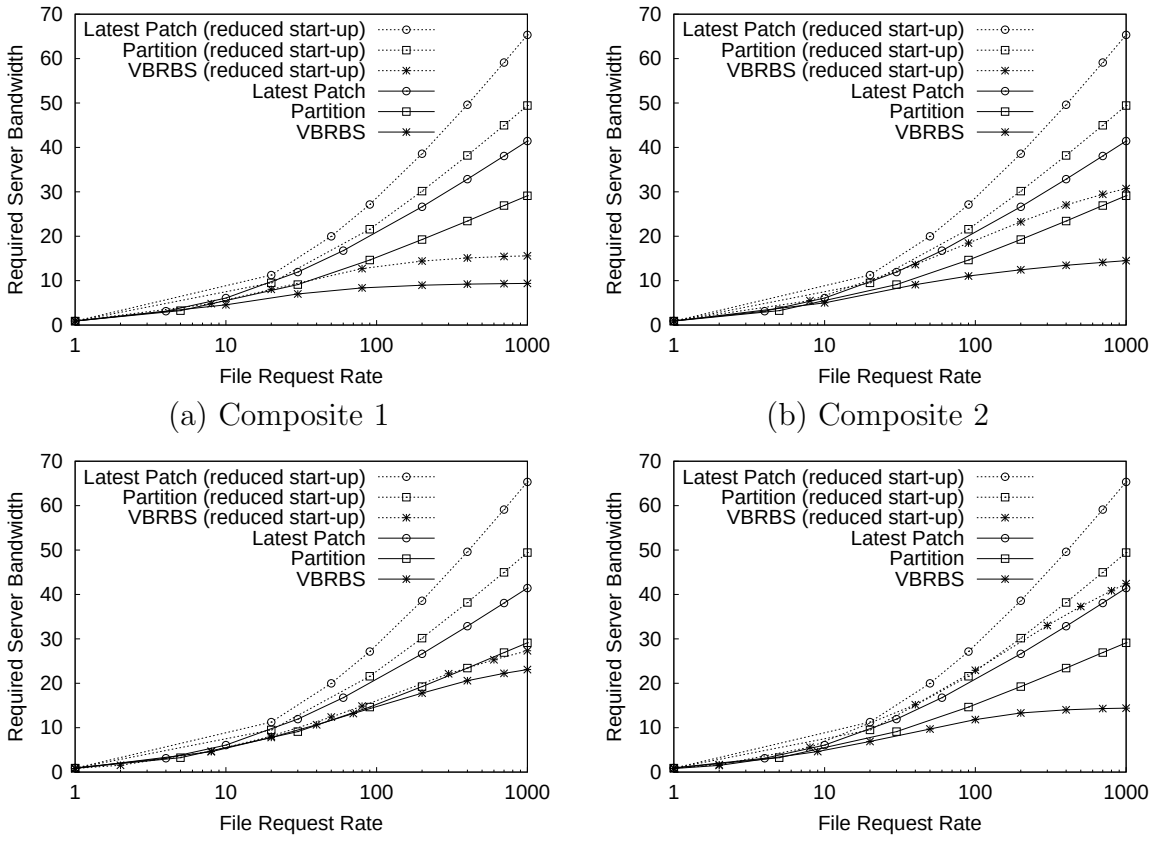
In this section, the performance of VBRBS assuming unlimited client buffering capacity is compared against the straightforward approach that applies the bandwidth skimming protocol to a previously fully smoothed VBR file, in Section 4.5.1 and Section 4.5.2, and in Section 4.5.3 the two variants of finite buffer VBRBS are compared. The performance metric is the required server bandwidth in units of the file's average bit rate.

### 4.5.1 Impact of File Request Rate

Figures 4.12 compares the server bandwidth requirements of VBRBS, *latest patch*, and *partition*, as a function of the normalized file request rate $N$. The client receive bandwidth (*crb*) is fixed at 1.2 times the file's average bit rate. The client buffer space is assumed large enough to allow full smoothing of the VBR file.

Six curves are depicted is in each graph. The curves labelled *Latest Patch* and *Partition* show the required server bandwidth when applying these techniques to deliver a VBR file that has been fully smoothed to the file's average bit rate, with some start-up delay, if required. The curve labelled *VBRBS* shows the server bandwidth required when applying VBRBS to the unsmoothed VBR file with the same start-up delay as required by full smoothing. For the curves labelled *Latest Patch (reduced start-up)* and *Partition (reduced start-up)*, the VBR file is fully smoothed to a rate that is equal to the minimal rate that can reduce the start-up delay to zero, or the file's average bit rate plus one-half of the amount by which the client receive bandwidth exceeds that rate, whichever is less. This choice of smoothing rate attempts to achieve a compromise between using extra bandwidth for stream merging and using it to reduce the start-up delay. Whatever start-up delay is required for *Latest Patch (reduced start-up)* and *Partition (reduced start-up)* is added to the VBR file delivered by VBRBS, for the curves labeled *VBRBS (reduced start-up)*.

Figure 4.12 illustrates that these scalable delivery protocols for delivering VBR media are scalable in that their server bandwidth requirements grow logarithmically




(a) Composite 1 (b) Composite 2

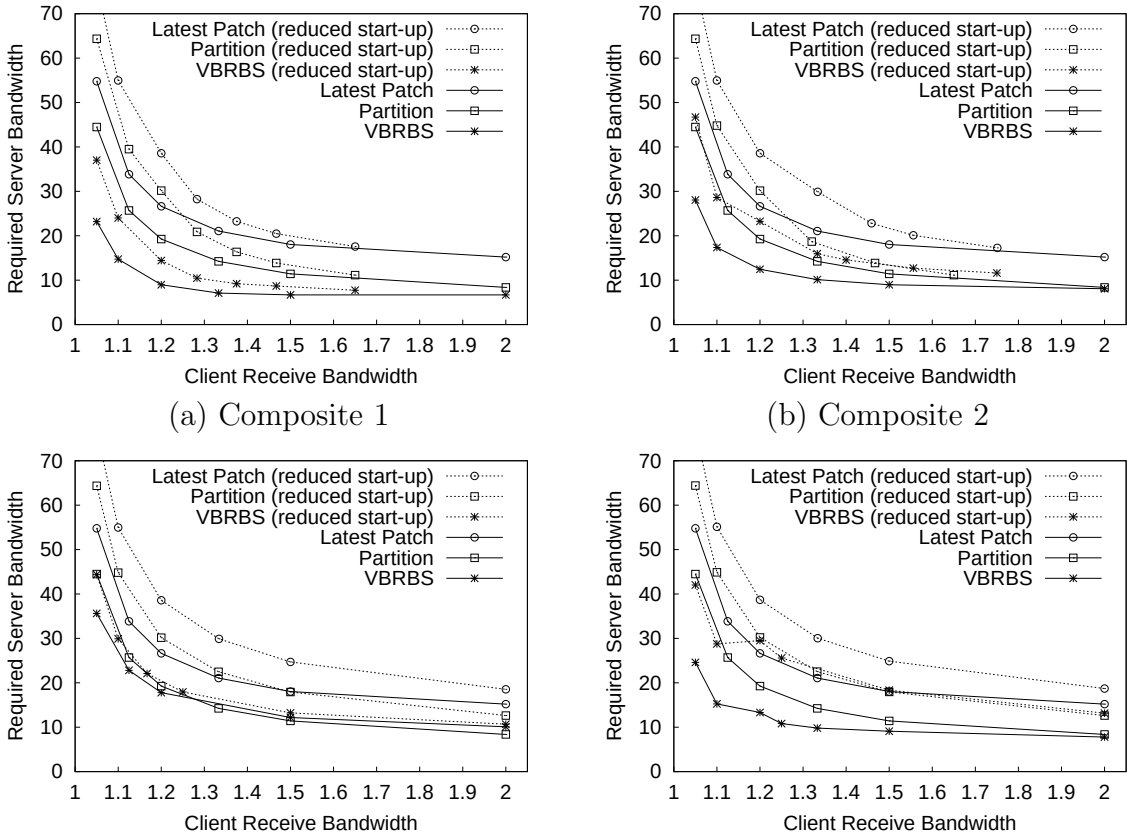
(c) Asterix (d) Dinosaur

Figure 4.12: Impact of File Request Rate
($crb = 1.2$)

with the file request rate. Compared with *latest patch*, VBRBS always consumes substantially less server bandwidth. VBRBS outperforms (sometimes considerably) *partition* when streaming hot objects (i.e., $N \geq 100$). In other cases, these two protocols have similar server bandwidth requirements. Moreover, there has much less scope for reducing required server bandwidth by use of VBRBS for Asterix with smoothing to the file's average bit rate (the line labelled *VBRBS* in Figure 4.12(c)), and Dinosaur with smoothing to 1.2 times the average rate (the line labelled *VBRBS reduced start-up delay* in Figure 4.12(d)), than in the other cases considered in the figure, which is consistent with the results in Figure 4.6. In general, the extent to which VBRBS reduces server bandwidth usage is heavily dependent on the rate variability in the VBR file, especially that in the beginning portions of the file.

### 4.5.2 Impact of Client Receive Bandwidth

Figure 4.13 compares the required server bandwidth of VBRBS, *latest patch*, and *partition*, as a function of the client receive bandwidth (*crb*). (Note again that the *crb* is in units of the file's average bit rate.) The file request rate is fixed at 200. The results shown in these figures are consistent with the conclusions drawn from Figures 4.12 concerning the relative performance of VBRBS, *latest patch*, and *partition*. Moreover, the figures show that VBRBS can substantially outperform partition when the client receive bandwidth is only a small percentage (less than 50%) more than the file's average bit rate. Note that there is occasional raggedness in the VBRBS curves. These irregularities are caused by the variations in start-up delays when a VBR file is smoothed at various rates.




(a) Composite 1

(b) Composite 2

(c) Asterix

(d) Dinosaur

Figure 4.13: Impact of Client Receive Bandwidth
($N = 200$)

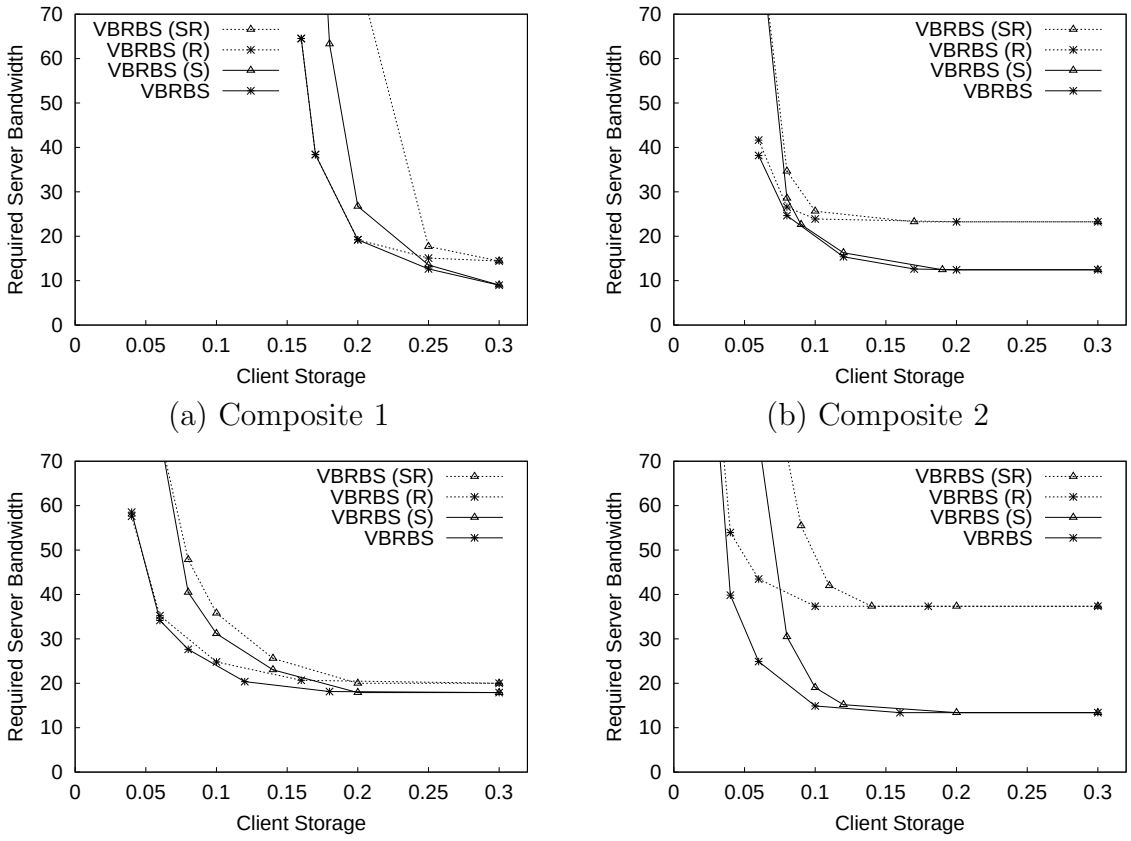
### 4.5.3 Impact of Client Buffer Capacity

Figure 4.14 compares the server bandwidth requirements of VBRBS and *VBRBS with smoothing* (labelled as VBRBS(S)), as a function of client buffer capacity. The client buffer capacity is in units of the file size. The client receive bandwidth (*crb*) is fixed at 1.2 times the file's average bit rate, and the file request rate is 200. For VBRBS and *VBRBS with smoothing*, the same start-up delay that results from fully smoothing the VBR file to its average bit rate is used. For the other two curves VBRBS (R) and VBRBS (SR), where R stands for *reduced start-up delay*, the minimum start-up delay (possibly equal to 0) that can be achieved by work-ahead smoothing (i.e., the start-up delay when fully smoothed at the client receive bandwidth) is used. Figure 4.14 shows that reducing the start-up delay increases the server bandwidth usage for streaming VBR files, as expected. Moreover, there is a tradeoff between the server bandwidth efficiency and the smoothness of the transmitted stream since there is a substantial increase in the server bandwidth usage when using optimally smoothed delivered streams.

## 4.6 Summary

This chapter has considered scalable on-demand streaming of VBR-encoded files. Work-ahead smoothing is a key technology for reducing the rate variability of VBR streams. However, in the context of on-demand streaming, this technique fundamentally conflicts with scalable delivery protocols, and can increase the server and network bandwidth usage. Previous research has investigated the impact of work-ahead smoothing on the bandwidth usage of periodic broadcast protocols for VBR media.

This chapter first derived a lower bound on the server bandwidth requirement for scalable on-demand streaming of VBR media. Using this lower bound analysis technique, it was found that the straightforward approach that directly applies scalable delivery protocols designed for CBR media to a fully-smoothed VBR file can consume substantially more server bandwidth than delivering the VBR file as is.




(a) Composite 1 (b) Composite 2

(c) Asterix (d) Dinosaur

Figure 4.14: Impact of Client Buffer Capacity
($N = 200$, $crb = 1.2$)

Motivated by this result, a new immediate service protocol, VBR bandwidth skimming (VBRBS), was designed. VBRBS delivers fixed-rate streams (when sufficient client storage is available) over the network, yet exploits the knowledge of the bit rate profile of a VBR file. Simulation results showed that VBRBS can considerably reduce the server bandwidth usage in comparison to that of the straightforward approach.

# Chapter 5

# Scalable On-demand Streaming of Non-linear Media

The multimedia files delivered in current streaming applications are *linear* in that they contain a single temporal sequence of media units. All clients requesting the same file receive all or a subsequence of the same sequence. Network delivery, in contrast to previous delivery methods such as TV broadcasting, may efficiently support a new type of media proposed here, in which clients can dynamically select among parallel sequences of media units at designated branch points. Examples of possible applications of this new *non-linear* media type are pick-your-own-ending movies, analogous to pick-your-own-ending children's books, or interactive on-line courses where students can select materials of their specific needs.

Non-linear media can be leveraged by future streaming applications to provide more personalized and customized content delivery services. For example, individual viewers of a future movie-on-demand system watching the same movie could select different story lines at pre-defined branch points; different clients visiting a future video-news web site could receive the same national news, followed by different provincial news and local news corresponding to the clients' respective geographic locations.

Realizing these advanced delivery services poses many challenges to scalable delivery protocols, which achieve substantial server and network bandwidth efficiency by broadcasting or multicasting a server stream to multiple clients that have made closely-spaced requests for the same file. Since clients requesting the same non-linear media file may receive different content as a result of their selections at branch points, there may be fewer opportunities for sharing streams among clients. One objective

of this study is to understand the extent to which scalable delivery protocols can still yield substantial bandwidth reductions, when applied to non-linear media, and the conditions under which these bandwidth reductions become negligible.

In order for clients that have made closely-spaced requests for a file to share the reception of the same server stream, scalable delivery protocols require that some clients receive and buffer file data ahead of its playback time. For non-linear media, since it may not be known what choice a client will make at a branch point until playback at the client reaches the branch point, such advance buffering may result in clients receiving data that is on different branches of the media file than those selected. Therefore, there is a tradeoff between reducing server-side bandwidth usage by aggressive "snooping" (i.e., listening to shared multicast streams delivering data beyond the current playback point) and decreasing client-side overhead by carefully restricting snooping so as to ensure that only useful data is received. The second objective of this study is to understand this tradeoff by quantifying both the bandwidth reduction benefit and the data overhead penalty for various scalable delivery approaches.

If the server had some *a priori* knowledge of client path selection, it could make better decisions as to which streams clients should listen to and on aggregating clients into groups. Such knowledge can be either independent of the specific client (i.e., overall branch selection probabilities or client-specific. The final objective of this study is to understand how much advance knowledge can help to reduce the bandwidth usage and data overhead of scalable delivery approaches.

Note that this study focuses on the *server bandwidth* savings from use of scalable on-demand streaming techniques. The potential reductions in network bandwidth requirements can be analyzed using the techniques developed for linear media in Chapter 3.

The remainder of this chapter is organized as follows. Section 5.1 describes non-linear media models. Section 5.2 proposes a number of scalable delivery approaches for non-linear media, and develops tight lower bounds on the required server bandwidth of these approaches and expressions for the associated client data overhead.

Section 5.3 presents the design of a number of practical scalable delivery protocols, and evaluates their server bandwidth requirements against the lower bound. As a proof-of-concept, Section 5.4 discusses the development of a prototype non-linear media streaming system, and preliminary experimentation with the system.

## 5.1 Non-linear Media Models

### 5.1.1 Non-linear Media Structures

A reasonably general structure for non-linear media is a tree structure, as illustrated in Figure 5.1. The root node represents the start of the file, the intermediate nodes correspond to branch points, and the leaf nodes signal the end of the file. The tree links represent the media portions that make up the file. It is assumed that at a branch point, clients make navigation decisions early enough to ensure jitter-free playback of the next media portion following the end of the current media portion, but close enough to the branch point so that the gap between when the decision is made and when the playback of the next portion starts can be ignored in the following analyses. It is also assumed that there is a common initial portion of the media file that all clients receive. Each path from the root to a leaf node defines a *complete playback path* of the non-linear media file.

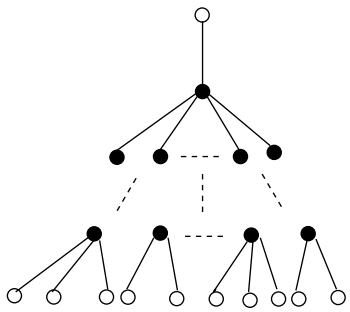


Figure 5.1: Tree Structure for Non-linear Media

A more general structure for non-linear media would be a directed acyclic graph (DAG) where multiple playback paths may converge. The DAG structure can be

further extended to a general graph structure where loops may exist, and where a playback may start at any node and proceed as graph links permit.

In this chapter, for clarity, all of the numeric results presented are for balanced binary tree-structured non-linear media, with all media portions having equal playback lengths and all clients requesting complete playbacks. Nevertheless, the analysis for the minimum required server bandwidth in Section 5.2 is developed for arbitrary tree structures, and can be further generalized. For the non-linear media scalable delivery protocols devised in Sections 5.3, the immediate service protocols can be applied to non-linear media having a general graph structure, while the periodic broadcast protocols are applicable to directed acyclic graphs in which the lengths of the paths to any video portion with multiple parents are identical, and to general tree structures.

It is also assumed that the non-linear media content is constant bit rate. Extension to variable bit rate non-linear media is possible, using similar approaches as for variable bit rate linear media as described in Chapter 4 and elsewhere [100, 86, 84].

### 5.1.2 Client Branch Selections

A key issue in modeling non-linear media is how to model the access probabilities of the various media portions. In this study, several popularity models have been explored for tree-structured non-linear media. These models all use the Zipf distribution.

The Zipf distribution is defined as follows. Given $W$ objects whose access probabilities follow a Zipf distribution, the probability of choosing the $i^{th}$ most popular object is $\frac{1}{i^{\alpha} \sum_{j=1}^{W} \frac{1}{j^{\alpha}}}$, where $\alpha \geq 0$. The parameter $\alpha$ is called the *skew factor*. When $\alpha = 0$, the $W$ objects are uniformly selected with a probability of $1/W$. As $\alpha$ increases, more skewness in the accesses to the objects is introduced. In this chapter, if not otherwise stated, $\alpha = 1$ is assumed.

The first popularity model assumes that clients select among the complete playback paths of a non-linear media file according to a Zipf distribution. Since each

complete playback path corresponds to a different leaf portion, the selection probabilities of the leaf portions will be a Zipf distribution. The most popular leaf portion is randomly selected and assigned the corresponding Zipf popularity, then the next most popular portion is randomly selected, and so on. The selection probability of each interior portion is computed by summing those of its children, starting from the portions whose children are leaves and progressing to the root. Note that the computed selection probability of the common initial portion will be equal to 1.

The second popularity model also assumes that the leaf portions are selected according to a Zipf distribution. However, this model assigns selection probabilities to the leaf portions from left to right in descending probability order, resulting in the leftmost leaf portion having the highest selection probability and the rightmost leaf portion having the lowest selection probability. The selection probabilities of all interior media portions are computed in the same way as in the first model.

The last popularity model considered here assigns relative selection probabilities to the media portions following each branch point according to a Zipf distribution. The selection probability of the common initial portion is set to 1, and then the selection probability of each subsequent portion is computed from that of its parent and its relative selection probability among its siblings.

Compared to the first popularity model, the second and the last models result in non-linear media trees with more skew in the selection probabilities of sibling media portions near the root of the tree and less skew among those close to the leaves. However, as will be seen in Figure 5.5, in most cases these models yield fairly similar results. For simplicity, in this chapter, if not otherwise stated, all numerical results are obtained using the first popularity model.

### 5.1.3 A Sample Non-linear Media Tree

Figure 5.2 shows a non-linear media file structured as a balanced binary tree of height 3 (following the standard terminology for tree data structure, the root portion is considered height *zero*) and selection probabilities chosen according to the

first popularity model described in the previous section. Each solid black circle represents a branch point. Each tree link corresponds to a media portion with the label indicating its selection probability. The selection probabilities of leaf portions follow a Zipf distribution with $\alpha = 1$. The selection probabilities of other media portions are computed by working up the tree from the leaves as described in the previous section. The figure also shows a complete playback path, as denoted by thick dark lines, which is selected in about 4.6% of the client playbacks. This unpopular playback path actually contains two very popular media portions at the first and the second levels of the tree, which are selected in about 56% and 41% of all client playbacks, respectively.

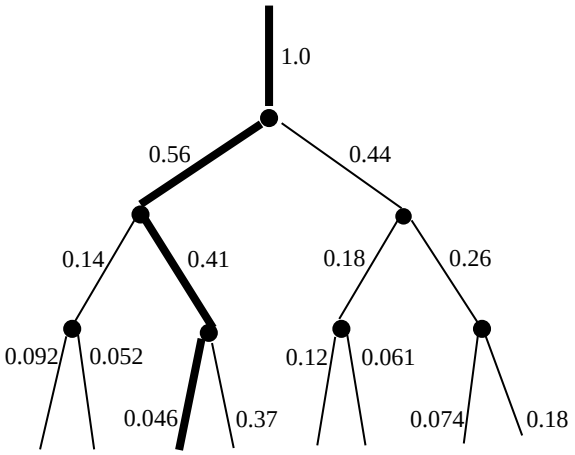



Figure 5.2: A Sample Non-linear Media Tree

### 5.1.4 Server Knowledge of Client Path Selection

An issue related to that of modelling access probabilities, is that of the possible *a priori* knowledge of client path selection. This study considers the following three scenarios.

- There is no *a priori* knowledge, that is, it is not known which media portion a client will playback next until the client makes the selection at a branch point. Prior to this point, it can be assumed only that all possible choices are equally

likely. Section 5.2.1 will show that even in this case, scalable delivery protocols can still achieve substantial server bandwidth reductions from unicast delivery.

- The overall average client selection probability for each media portion is known. Using this information, at each branch point it could be predicted that a client will choose the most popular branch. It can be seen that in this case, the probability of an incorrect prediction is equal to one minus the relative selection frequency of the most popular branch. If client accesses to sibling media portions are very skewed at a branch point, i.e., the relative selection frequency of the most popular branch is close to 1, predicating that all clients will take that branch works very well; otherwise, this policy may result in a high rate of incorrect predictions. In Section 5.2.2, the server bandwidth analyses of several scalable delivery approaches based on such predictions indicate that they can be quite inefficient.

- Client-specific advance knowledge of path selections is available, for example, by measuring the previous behaviour of clients either individually or in aggregation according to some client classification. Based on this information, a playback path could be predicted for any particular client. In Section 5.2.2, a simple model of client-specific path selection accuracy will be defined, and a scalable delivery approach utilizing this model will be described and its server bandwidth efficiency analyzed.

## 5.2 Analysis for Minimum Server Bandwidth

This section discusses a number of multicast-based scalable delivery approaches for non-linear media, and presents analysis results for the minimum required server bandwidth and the associated client data overhead of each of these approaches. Section 5.2.1 focuses on the potential for server bandwidth reduction by use of scalable delivery, and derives a tight lower bound on the required server bandwidth of any scalable delivery approaches. If the server had no *a priori* knowledge of

client path selection, achieving this lower bound would result in high client data overhead. In Section 5.2.2, several scalable delivery approaches that can reduce client data overhead at some cost of server bandwidth usage are developed and analyzed. Table 5.1 summarizes the notation used in the following analyses.

Table 5.1: Notation for Non-linear Media Scalable Delivery Approaches

| Symbol | Definition |
|---|---|
| $V$ | Number of portions of the non-linear media file |
| $T$ | Complete path playback time |
| $T_i$ | Playback time of the $i^{th}$ portion; portions are numbered top-down from left to right, with the common initial portion numbered as portion 1 |
| $t_i$ | Relative start time of portion $i$ ($t_1 = 0$) |
| $p_i$ | Probability the selected path includes portion $i$ |
| $\lambda$ | File request rate |
| $\lambda_i$ | Client request rate for portion $i$ ($\lambda_i = p_i\lambda$) |
| $N$ | Normalized file request rate; average number of client requests during a playback time ($N = \lambda T$) |
| $N_i$ | Average number of client requests for portion $i$ during time $T_i$ ($N_i = \lambda_i T_i$) |
| $d$ | Maximum client start-up delay |
| $\alpha$ | Parameter of Zipf distribution (popularity of the $j^{th}$ most popular item $\propto 1/j^{\alpha}$) |
| $B_{min}$ | Required server bandwidth lower bound, in units of the file playback bit rate |
| $O_{min}$ | Average client data overhead, in units of the amount of file data on a complete playback path |

### 5.2.1 Potential for Scalable Delivery

This section derives a tight lower bound on the minimum server bandwidth usage for scalable delivery of non-linear media. In the absence of *a priori* knowledge of client path selection, achieving this bound would require use of *unrestricted snoop-ahead*. *Unrestricted snoop-ahead* maximizes possible sharing of server transmissions among clients accessing the same non-linear media file. With this policy, a client listens to any multicast of data from media portions in the subtree below its current playback position, in other words, to any multicast of data that it might need in the future.

#### 5.2.1.1 Minimum Required Server Bandwidth

The lower bound on the required server bandwidth for scalable delivery of non-linear media can be derived similarly as that for linear media, as was discussed in Section 2.4. For a non-linear media file, the analysis considers a client whose request for the file is made at time $t$, and that will include the $i^{th}$ media portion in its playback path. Consider an infinitesimally small segment of data $\Delta x$ at position $x$ in the $i^{th}$ media portion. This data $\Delta x$ is at an overall playback position $t_i + x$ from the beginning of the non-linear media file. In order to guarantee a deterministic start-up delay $d$, $\Delta x$ must be multicast to the client no later than time $t+d+(t_i+x)$. During the interval $[t, t+d+(t_i+x)]$, any new clients can share this multicast. After time $t+d+(t_i+x)$, the next client that requests the file and whose playback path will include the $i^{th}$ media portion will trigger a new multicast to be scheduled. Assuming that client request arrivals are Poisson, the average duration from time $t+d+(t_i+x)$ to the next such client request $1/\lambda_i$. Therefore, the minimum frequency at which $\Delta x$ must be multicast is $\dfrac{1}{d+t_i+x+1/\lambda_i}$, which yields a tight lower bound on the required server bandwidth of:

$$\begin{aligned} B_{min} &= \sum_{i=1}^{V} \int_0^{T_i} \frac{1}{d+t_i+x+1/\lambda_i} dx \\ &= \sum_{i=1}^{V} \ln\left(\frac{N_i}{N_i \frac{d+t_i}{T_i}+1}+1\right). \end{aligned} \tag{5.1}$$

For comparison purposes, this section also considers two delivery approaches that simply apply scalable delivery techniques for linear media to non-linear media. In the first approach, *portion*, each portion of a non-linear media file is treated as a separate linear media file. In the second approach, *path*, each complete playback path of a non-linear media file is delivered to clients as if it were a separate linear media file.

The lower bound on the required server bandwidth for the *portion* approach can be derived by applying the analysis technique discussed in Section 2.4 to each media portion, yielding:

$$B_{min}^{portion} = \sum_{i=1}^{V} \int_0^{T_i} \frac{1}{d_i + x + 1/\lambda_i} dx$$
$$= \sum_{i=1}^{V} \ln\left(\frac{N_i}{N_i \frac{d_i}{T_i} + 1} + 1\right). \tag{5.2}$$

Here $d_1$ is equal to the start-up delay $d$. The terms $d_i$ for $i > 1$ model a scenario in which clients are required to select the $i^{th}$ portion some time $d_i$ before they start the playback of that portion; or alternatively, in which there is an interruption of duration $d_i$ between the end of the playback of portion $i$'s parent portion and the start of the playback of portion $i$.

The *path* approach requires that client path selections are known *a priori*. The probability with which the linear file corresponding to a complete playback path is selected by a client is equal to the selection probability of the leaf portion that ends the path. The lower bound on the required server bandwidth for this approach can be computed by applying the analysis technique discussed in Section 2.4 to each playback path, which yields:

$$B_{min}^{path} = \sum_{i \in \mathcal{L}} \int_0^{T} \frac{1}{d + x + 1/\lambda_i} dx$$
$$= \sum_{i \in \mathcal{L}} \ln\left(\frac{p_i N}{p_i N \frac{d}{T} + 1} + 1\right), \tag{5.3}$$

where $\mathcal{L}$ denotes the set of indices of the leaf portions, and for notational convenience it is assumed that each complete playback path has the same playback time $T$.

Figure 5.3 shows the above lower bounds on the required server bandwidth as a function of the normalized file request rate $N$. For comparison, the minimum required server bandwidth for scalable delivery of linear media, computed from expression (2.2), is also shown. The non-linear media file structure is a balanced binary tree of height 3, as in Figure 5.2. In this tree and all the other balanced

binary trees used in the experiments, it is assumed that all media portions have equal length and all complete playback paths have equal playback duration. The results are for immediate service (i.e., the start-up delay $d = 0$), and the first of the popularity models presented in Section 5.1.2 (in which leaf selection probabilities are chosen randomly according to a Zipf distribution and selection probabilities of interior nodes are computed from those of their children) with skewness factor $\alpha = 1$.

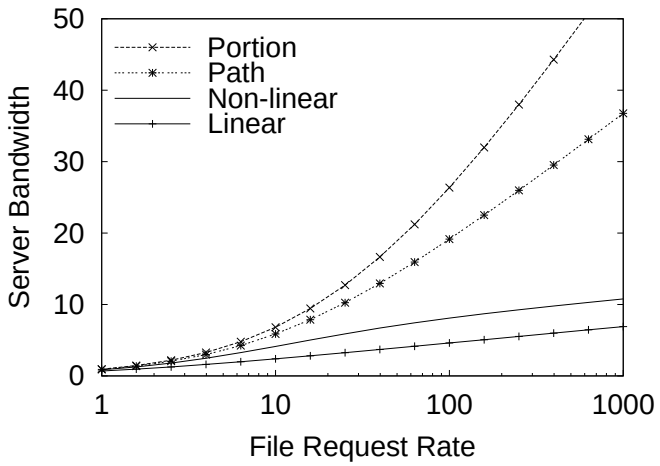



Figure 5.3: Server Bandwidth for Non-linear Media
(balanced binary tree with height 3, $\alpha = 1$, $d = 0$)

This figure shows that scalable delivery has a large potential for server bandwidth reduction from that of unicast delivery, whose server bandwidth requirement is equal to the normalized client request rate $N$. This is particularly the case when the non-linear media structure is recognized by the delivery protocol rather than simply treating each portion or playback path as a separate linear media file.

Figure 5.4 illustrates the impact of the height of the non-linear media tree structure on the server bandwidth reduction potential of scalable delivery approaches. The results shown are for client request rate $N$ of 1000, immediate service (i.e., start-up delay $d = 0$), and leaf selection probabilities chosen randomly according to a Zipf distribution with $\alpha = 1$.

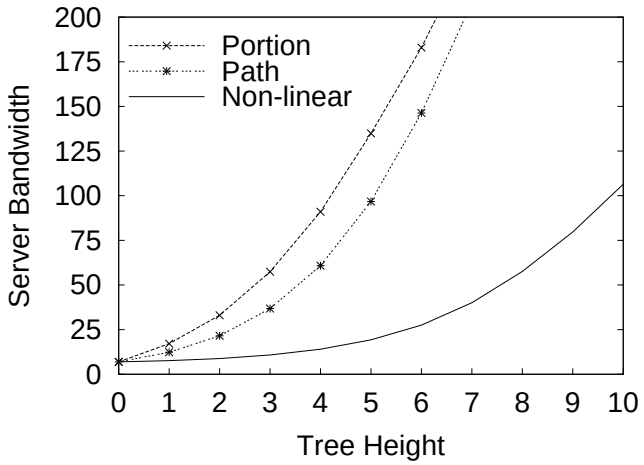



Figure 5.4: Impact of Tree Height
(balanced binary tree, $\alpha = 1$, $N = 1000$, $d = 0$)

Figure 5.4 shows that as the non-linear media tree structure becomes deeper, the potential benefit of scalable delivery over unicast delivery decreases. This is because as the height of a non-linear media tree increases, the number of media portions and the number of possible playback paths increase exponentially, and furthermore the length of a media portion decreases relative to the total length of a path, resulting in less potential for sharing of the transmissions of a particular media portion or playback path. For the same reason, an increase in the branching factor at each branch point will also decrease the potential benefit of scalable delivery. However, even for a tree of height ten and over one thousand possible playback paths, the minimum required server bandwidth with scalable delivery is still an order-of-magnitude lower than that with unicast delivery, assuming immediate service and $N = 1000$ as in the figure. Much of these potential bandwidth savings come from sharing the transmissions of media portions with relatively high selection probabilities, which reside either close to the root or along popular playback paths.

Figure 5.5 shows the impact of using alternative popularity models, as discussed in Section 5.1.2, on the potential server bandwidth reduction, as a function of the height of the non-linear media tree structure. The client request rate is fixed at 1000, and immediate service (i.e., start-up delay $d = 0$) is assumed. For all popularity

models, the skewness factor $\alpha$ is set to 1. The figure shows that these popularity models give quite similar results, with significant differences only for very deep trees. Subsequent results assume the first popularity model from Section 5.1.2, labelled *Zipf on Leaves Randomly* in Figure 5.5.

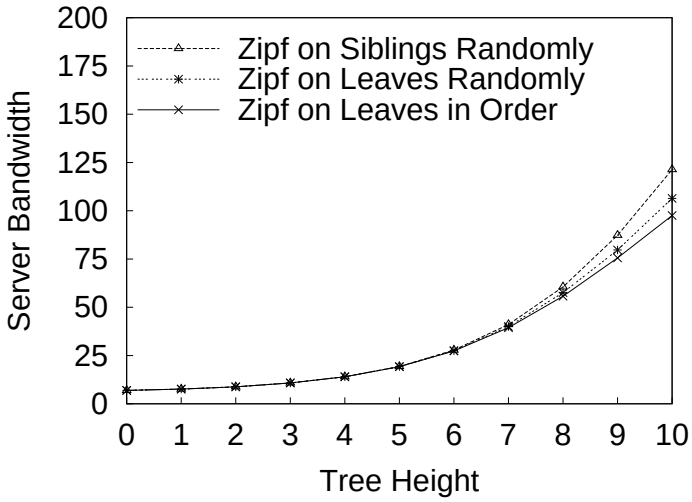



Figure 5.5: Impact of Popularity Model
(balanced binary tree, $\alpha = 1$, $N = 1000$, $d = 0$)

#### 5.2.1.2 Client Data Overhead

Without *a priori* knowledge of client path selection, achieving the lower bound of equation (5.1) would require use of *unrestricted snoop-ahead*, in which a client listens to all multicasts of data on playback paths the client may take. Only after a client selects a branch at a branch point, can it stop listening to multicasts of data from the subtrees rooted at the other branches. As a result, clients may receive a large amount of data that they do not use. To measure this overhead, the *average data overhead* is defined as the average total amount of data that a client receives from media portions that are not on the client's chosen playback path, expressed in units of the amount of data on a complete playback path. The average amount of data that a client receives from such a portion can be computed as the product of the data rate of multicasts from the portion and the duration over which the client listens to these multicasts.

It can be easily seen that for the *portion* approach, there is no data overhead, regardless of whether the system has or does not have *a priori* knowledge of client path selection. For the *path* approach, since the client path selection must be known before reception begins, there is again no data overhead. With advance knowledge of client path selection, achieving the lower bound of equation (5.1) also requires no data overhead. Without advance knowledge, on the other hand, *unrestricted snoop-ahead*, assuming that all complete playback paths contain the same amount of data for notational simplicity, consumes a minimum average client data overhead of:

$$O_{min}^{unrestricted} = \frac{\sum_{i \in \mathcal{L}} p_i \sum_{j \in \overline{\mathcal{A}}(i)} \left( d + \sum_{k \in \mathcal{A}(i,j)} T_k \right) \ln \left( \frac{N_j}{N_j \frac{d+t_j}{T_j} + 1} + 1 \right)}{T}, \tag{5.4}$$

where $\mathcal{L}$ denotes the set of indices of the leaf portions, $\overline{\mathcal{A}}(i)$ denotes the set of indices of those portions that are not ancestors of portion $i$ (excluding $i$ itself), and $\mathcal{A}(i,j)$ denotes the set of indices of those portions that are ancestors of both portion $i$ and portion $j$. The denominator of equation (5.4) is the amount of data on a complete playback path. In the numerator of equation (5.4), $\ln \left( \frac{N_j}{N_j \frac{d+t_j}{T_j} + 1} + 1 \right)$ is the minimum data rate (bandwidth) of multicasts from portion $j$, which is computed from $\int_0^{T_j} \frac{1}{d + t_j + x + 1/\lambda_j} dx$, where $\frac{1}{d + t_j + x + 1/\lambda_j}$ is the minimum rate at which an infinitesimally small portion of data $\Delta x$ at time offset $x$ from the start time of portion $j$ (i.e., $t_j$) is multicast. Also in the numerator of equation (5.4), $\left( d + \sum_{k \in \mathcal{A}(i,j)} T_k \right)$ is the duration during which a client listens to multicasts of data from portion $j$. In total, the numerator of equation (5.4) reflects the fact that using the *unrestricted snoop-ahead* approach, if leaf portion $i$ is on the chosen playback path of a client (with probability $p_i$), the data that the client receives from the portions that are not the ancestors of portion $i$ is the overhead data. Multicasts of data from such a portion $j$ are listened to by the client during the start-up delay $d$, and prior to the branch point at which its path and the path including portion $j$ diverge.

Figure 5.6 shows the minimum average client data overhead incurred by the *unrestricted snoop-ahead* approach as a function of the normalized file request rate $N$ for various non-linear media tree structure heights, assuming immediate service (i.e., $d = 0$). The client data overhead eventually flattens out as the client request rate increases. This is because the data rate of multicasts from portion $j$ for $j > 1$ (i.e., $\ln\left(\frac{N_j}{N_j\frac{d+t_j}{T_j}+1}+1\right)$) has finite asymptote. (For tree heights of greater than 6, note that this flattening out occurs for higher request rates than those considered in the figure.) Moreover, for fixed client request rate, as the height increases the client data overhead also initially increases but will level off, and for $d = 0$ eventually decrease, since the resulting increase in the number of possible paths results in a small proportion of the multicasts that are of data from below the current play point in the media tree. Finally, the figure shows that the data overhead when clients snoop on all portions that could still be on their eventual path can be significant, particularly when the tree height is greater than 4 and the normalized request rate is greater than 100.

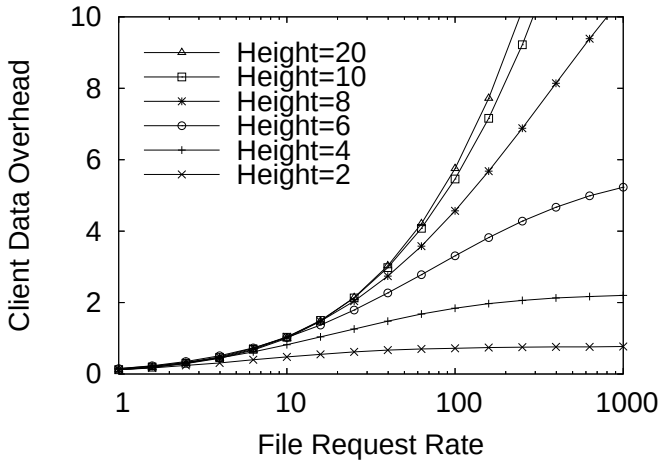



Figure 5.6: Client Data Overhead for *Unrestricted Snoop-ahead*
(balanced binary tree, $\alpha = 1$, $d = 0$)

### 5.2.2 Restricted Snoop-ahead Approaches

The *unrestricted snoop-ahead* approach can result in high client data overhead, as seen in Figure 5.6. This section discusses a number of restricted snoop-ahead approaches in which clients snoop on multicasts less aggressively, thus considerably reducing the average client data overhead, although at the cost of increased server bandwidth usage.

Snoop-ahead can be restricted based on a number of criteria, including: 1) distance of the multicast data from the current playback position; 2) overall portion selection probabilities; and 3) client-specific path prediction based on client past behaviour, client classification, and/or advance selection. Some representative restricted snoop-ahead approaches are described below. Their server bandwidth requirements and client data overheads are derived and compared against those of the *unrestricted snoop-ahead* approach.

#### 5.2.2.1 Distance-based Restricted Snoop-ahead

##### The *Next* Approach

A simple distance-based restricted snoop-ahead approach is *next*. In this approach, a client receives multicasts of data from the current portion (but ahead of the client's current playback position) and the portions following the next branch point. For instance, during playback of a balanced binary tree-structured non-linear media file, if the client's current playback position is within a leaf portion, the client will be listening to multicasts of data only from that portion. If the client's current playback position is within any one of the non-leaf portions, the client will be listening to three media portions, including the current portion and its two children portions. During the initial start-up delay, if any, clients listen only to multicasts of data from the initial root portion.

The minimum server bandwidth required by the *next* approach can be derived as:

$$\begin{aligned} B_{min}^{next} &= \int_0^{T_1} \frac{1}{d + x + 1/\lambda} dx + \sum_{i=2}^{V} \int_0^{T_i} \frac{1}{T_{a(i)} + x + 1/\lambda_i} dx \\ &= \ln\left(\frac{N_1}{N_1 \frac{d}{T_1} + 1} + 1\right) + \sum_{i=2}^{V} \ln\left(\frac{N_i}{N_i \frac{T_{a(i)}}{T_i} + 1} + 1\right), \end{aligned} \tag{5.5}$$

where $a(i)$ denotes the index of the immediate ancestor (i.e., the parent) of portion $i$.

The minimum average client data overhead with this approach is given by:

$$O_{min}^{next} = \frac{\sum_{i=2}^{V} p_i T_{a(i)} \sum_{j \in \mathcal{S}(i)} \ln\left(\frac{N_j}{N_j \frac{T_{a(j)}}{T_j} + 1} + 1\right)}{T}, \tag{5.6}$$

where $\mathcal{S}(i)$ denotes the set of indices of the siblings of portion $i$. For the common initial portion (i.e., portion 1), the *next* approach does not incur any overhead. For each sibling portion $j$ of a portion $i$ ($2 \leq i \leq V$) that is on a client's playback path (with probability $p_i$), all the data that the client receives from portion $j$ will be unused and thus overhead. The minimum data rate of multicasts from portion $j$ is $\ln\left(\frac{N_j}{N_j \frac{T_{a(j)}}{T_j} + 1} + 1\right)$, and the duration during which the client listens to such multicasts is $T_{a(i)}$.

### The *Nexttwo* Approach

Less restrictive distance-based approaches than the *next* approach allow clients to receive multicasts of data from two or more levels of media portions below the next branch point. For example, in the *nexttwo* approach, a client receives multicasts of data from the current portion the client is playing back and all portions no more than two branch points subsequent to this portion. During the start-up delay, if any, clients listen to multicasts of data from the root portion and the immediate children of that portion. The more portions potentially on its playback path that

a client snoops multicasts of data from, the closer the minimum server bandwidth requirement is to the lower bound, but the greater the client data overhead.

The minimum server bandwidth required by the *nexttwo* approach can be derived as:

$$\begin{aligned} B_{min}^{nexttwo} &= \int_0^{T_1} \frac{1}{d + x + 1/\lambda} dx + \sum_{i=2}^{V} \int_0^{T_i} \frac{1}{T_{g(i)} + T_{a(i)} + x + 1/\lambda_i} dx \\ &= \ln \left( \frac{N_1}{N_1 \frac{d}{T_1} + 1} + 1 \right) + \sum_{i=2}^{V} \ln \left( \frac{N_i}{N_i \frac{T_{g(i)} + T_{a(i)}}{T_i} + 1} + 1 \right), \end{aligned} \tag{5.7}$$

where $g(i)$ denotes the index of the grand-parent portion of portion $i$. If portion $i$ is an immediate child of the common initial portion, $T_{g(i)}$ is defined as equal to the start-up delay $d$.

The minimum average client data overhead incurred by this approach can be computed as:

$$O_{min}^{nexttwo} = \frac{\sum_{i=2}^{V} p_i \left( \left( T_{a(i)} + T_{g(i)} \right) \sum_{j \in \mathcal{S}(i)} r_j + T_{a(i)} \sum_{j \in \mathcal{S}(i)} \sum_{k \in \mathcal{C}(j)} r_k \right)}{T}, \tag{5.8}$$

where $r_i$ denotes the minimum data rate of multicasts from portion $i$, which is given by $\ln \left( \frac{N_i}{N_i \frac{T_{g(i)} + T_{a(i)}}{T_i} + 1} + 1 \right)$, and $\mathcal{C}(j)$ denotes the set of indices of the immediate child portions of portion $j$. If portion $j$ is a leaf portion, $\mathcal{C}(j)$ is the empty set. Expression (5.8) follows from the fact that the data overhead incurred if portion $i$ is chosen (with probability $p_i$) includes the data received from portion $i$'s sibling portions and their immediate child portions. The length of the period over which multicasts of data from portion $i$'s sibling portions are received is the sum of the playback lengths of portion $i$'s parent and grand-parent portions, and the duration over which the child portions are received is equal to the playback length of portion $i$'s parent portion.

The server bandwidth and the average client data overhead for distance-based approaches in which clients snoop on multicasts from media portions up to $k$ ($k > 2$)

branch points away from the current playback point can be derived in an analogous fashion as for the *next* and *nexttwo* approaches.

#### 5.2.2.2 Client Path Prediction Approaches

##### The *Popular Next* Approach

Client path prediction approaches can utilize either overall portion selection probabilities or client-specific path selection information. A simple approach that restricts snoop-ahead based on overall portion selection probabilities is *popular next*, where clients snoop on multicasts of data from the current portion and the most popular portion among all portions following the next branch point. If portion selection probabilities are sufficiently skewed, the *popular next* approach should be able to significantly reduce client data overhead, in comparison to the *next* approach, at the cost of increased server bandwidth usage. The minimum server bandwidth required by this approach can be derived as:

$$\begin{aligned} B_{min}^{popnext} &= \int_0^{T_1} \frac{1}{d + x + 1/\lambda} dx + \sum_{i \in \mathcal{P}} \int_0^{T_i} \frac{1}{T_{a(i)} + x + 1/\lambda_i} dx + \sum_{i \in \overline{\mathcal{P}}} \int_0^{T_i} \frac{1}{x + 1/\lambda_i} dx \\ &= \ln\left(\frac{N_1}{N_1 \frac{d}{T_1} + 1} + 1\right) + \sum_{i \in \mathcal{P}} \ln\left(\frac{N_i}{N_i \frac{T_{a(i)}}{T_i} + 1} + 1\right) + \sum_{i \in \overline{\mathcal{P}}} \ln\left(N_i + 1\right), \end{aligned} \quad (5.9)$$

where $\mathcal{P}$ denotes the set of indices of the media portions that are the most popular portions among their siblings (excluding the root portion), and $\overline{\mathcal{P}}$ denotes the set of indices of the remaining media portions (excluding the root portion). The server bandwidth requirement for each popular portion is the same as that with the *next* approach, while the server bandwidth requirement for each non-popular portion is equal to that with the *portion* approach.

In the *popular next* approach, if a client does select the most popular portion at the next level of the media tree after finishing the playback of the current portion $i$, no client data overhead will be incurred. If the client selects any one of the other non-popular media portions, the data overhead is the data the client has received from the most popular portion, yielding:

$$O_{min}^{popnext} = \frac{\sum_{i \in \overline{\mathcal{P}}} p_i T_{a(i)} \ln \left( \frac{N_{s(i)}}{N_{s(i)} \frac{T_{a(i)}}{T_{s(i)}} + 1} \right)}{T}, \tag{5.10}$$

where $s(i)$ denotes the index of the most popular sibling of portion $i$.

### The *Popular Path* Approach

In the *popular path* approach, clients listen to multicasts of data from portions on the most popular path in the sub-tree rooted at the current portion. During the initial start-up delay, if any, clients listen only to multicasts of data from the initial root portion. The minimum server bandwidth required by this approach is:

$$\begin{aligned} B_{min}^{poppath} &= \int_0^{T_1} \frac{1}{d + x + 1/\lambda} dx + \sum_{i \in \mathcal{P}} \int_0^{T_i} \frac{1}{\sum_{j \in \mathcal{U}(i)} T_j + x + 1/\lambda_i} dx + \sum_{i \in \overline{\mathcal{P}}} \int_0^{T_i} \frac{1}{x + 1/\lambda_i} dx \\ &= \ln \left( \frac{N_1}{N_1 \frac{d}{T_1} + 1} + 1 \right) + \sum_{i \in \mathcal{P}} \ln \left( \frac{N_i}{N_i \frac{\sum_{j \in \mathcal{U}(i)} T_j}{T_i} + 1} + 1 \right) + \sum_{i \in \overline{\mathcal{P}}} \ln \left( N_i + 1 \right), \end{aligned} \tag{5.11}$$

where for any popular portion $i$, $\mathcal{U}(i)$ denotes the set of indices of ancestors on the path back towards the root from portion $i$ (not including portion $i$ itself), up to and including the first portion that is not the most popular portion among its siblings. If there is no such portion on the path, the set includes the indices of all ancestor portions on the path back to and including the root. Since multicasts of data from non-popular portions are received by clients only after such portions are selected, the server bandwidth requirements for non-popular portions are the same

as with the *portion* approach. Clients listen to multicasts of data from each popular portion $i$, however, whenever the path from (but not including) the current portion to portion $i$ includes only portions that are the most popular among their siblings.

The minimum average client data overhead with this approach is given by:

$$O_{min}^{poppath} = \frac{\sum_{i \in \overline{\mathcal{P}}} p_i \left( \sum_{j \in \mathcal{U}(i)} T_j \right) \sum_{j \in \mathcal{D}(a(i))} \ln \left( \frac{N_j}{\sum_{k \in \mathcal{U}(j)} T_k} + 1 \right)}{T},$$

where $\mathcal{D}(a(i))$ denotes the set of indices of media portions on the most popular path down to a leaf from (but not including) the parent of portion $i$. Whenever a client selects a popular portion $i$, no data overhead is realized, since the client will not have been listening to multicasts of data from the siblings of $i$ or from portions in their associated subtrees. However, whenever a client selects a non-popular portion $i$ (with probability $p_i$), data received from the media portions along the most popular path from the current branch point to a leaf portion becomes overhead, as it will not be used. The time period over which the client has listened to multicasts of such data is $\sum_{j \in \mathcal{U}(i)} T_j$, and the minimum data rate of multicasts from such a portion $j \in \mathcal{D}(a(i))$ is $\ln \left( \frac{N_j}{N_j \frac{\sum_{k \in \mathcal{U}(j)} T_k}{T_j} + 1} + 1 \right)$.

### The *Prediction* Approach

Snoop-ahead can also be restricted using client-specific path predications based on client path behaviours, client classification, and/or clients' advance selections. In the prediction-based approach considered here (*prediction*), a client snoops on multicasts of data from all media portions along the predicted path from the client's current playback position to a leaf portion. Again, during the initial start-up delay, if any, clients listen only to multicasts of data from the initial root portion. This study considers a very simple prediction accuracy model that assumes that media

portions with conditional selection frequency (on reaching the respective branch point) greater than or equal to a parameter $f$ ($0 \leq f \leq 1$) are always correctly predicted, whereas media portions with selection frequency less than $f$ are never predicted. Note that as $f$ decreases, prediction accuracy increases, with perfect prediction achieved for $f = 0$. As $f$ increases, prediction accuracy decreases, and in the case of a binary tree structure, the approach is identical to *popular path* for $f = 0.5$, assuming no two siblings with identical popularities.

The prediction approach has a minimal server bandwidth requirement of:

$$B_{min}^{pred} = \int_0^{T_1} \frac{1}{d + x + 1/\lambda} dx + \sum_{i \in \mathcal{F}} \int_0^{T_i} \frac{1}{\sum_{j \in \mathcal{W}(i)} T_j + x + 1/\lambda_i} dx + \sum_{i \in \overline{\mathcal{F}}} \int_0^{T_i} \frac{1}{x + 1/\lambda_i} dx$$

$$= \ln\left(\frac{N_1}{N_1 \frac{d}{T_1} + 1} + 1\right) + \sum_{i \in \mathcal{F}} \ln\left(\frac{N_i}{N_i \frac{\sum_{j \in \mathcal{W}(i)} T_j}{T_i} + 1} + 1\right) + \sum_{i \in \overline{\mathcal{F}}} \ln\left(N_i + 1\right), \quad (5.12)$$

where $\mathcal{F}$ and $\overline{\mathcal{F}}$ denote the set of indices of the media portions (excluding the root portion) whose conditional selection frequencies are at least $f$, or less than $f$, respectively, and $\mathcal{W}(i)$ denotes the set of indices of ancestors on the path back towards the root from portion $i$ (not including portion $i$ itself), up to and including the first portion that is a member of set $\mathcal{F}$. If there is no such portion on this path, the set includes the indices of all ancestors on the path back to and including the root. Equation (5.12) can be understood by analogy to equation (5.11) of the *popular path* approach, noting that the sets $\mathcal{F}$ and $\overline{\mathcal{F}}$ are analogous to the sets $\mathcal{P}$ and $\overline{\mathcal{P}}$, respectively.

The minimum average client data overhead with this approach is given by:

$$O_{min}^{pred} = \frac{\sum_{i \in \overline{\mathcal{F}}} p_i \left(\sum_{j \in \mathcal{W}(i)} T_j\right) \sum_{l \in \mathcal{L}(\mathcal{S}(i))} \frac{p_l}{\sum_{m \in \mathcal{L}(\mathcal{S}(i))} p_m} \sum_{j \in \mathcal{D}(a(i),l)} B_{j\ min}^{pred}}{T}, \quad (5.13)$$

where $\mathcal{L}(\mathcal{S}(i))$ denotes the set of indices of video portions that are leaves in the collection of pruned subtrees rooted at siblings of $i$, where the pruning has removed

all video portions not in the set $\mathcal{F}$ and their descendents, $\mathcal{D}(a(i), l)$ denotes the set of indices of media portions on the path down to portion $l$ beginning from (but not including) the parent of portion $i$, $B_{j\ min}^{pred}$ denotes the minimum server bandwidth required for portion $j$, and where it is assumed that an incorrect path prediction is for each of the paths that could have been predicted (i.e., the paths containing only media portions in set $\mathcal{F}$) with probability proportional to its relative popularity, i.e., $\frac{p_l}{\sum_{m \in \mathcal{L}(\mathcal{S}(i))} p_m}$.

#### 5.2.2.3 Policy Comparisons

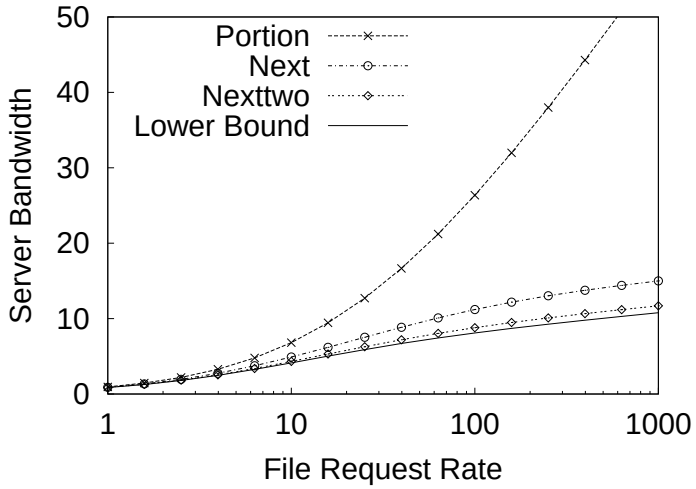
This section compares the server bandwidth requirements and client data overheads of the restricted snoop-ahead approaches described in the previous sections. Figure 5.7 shows the minimum required server bandwidth for distance-based (Figure 5.7(a)) and prediction-based (Figure 5.7(b)) restricted snoop-ahead approaches, as a function of the normalized file request rate $N$, assuming immediate service (i.e., $d = 0$) and a balanced binary tree media structure with height 3 and $\alpha = 1$. The lower bound from Section 5.2.1 is also shown.

Figure 5.7(a) shows that snooping on multicasts of data from the media portions immediately after the next branch point (i.e., *next*), in addition to multicasts from the current portion, can substantially reduce the server bandwidth requirement, to close to the minimum possible. Snooping on multicasts of data from media portions farther ahead (e.g., *nexttwo*) yields diminishing returns.

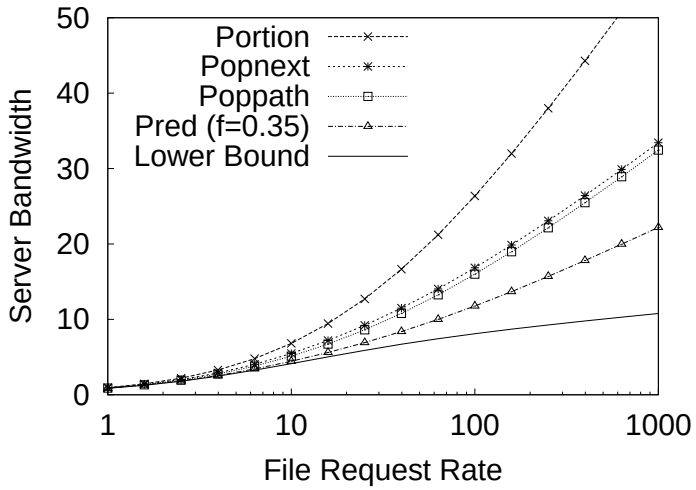
Figure 5.7(b) shows that *popular next* and *popular path*, which utilize overall portion selection probabilities for path prediction, often require a much higher server bandwidth than distance-based approaches. This poor performance can be explained by the fact that if the selection probabilities of siblings are similar (for example, as for the children of the root portion in the non-linear media tree shown in Figure 5.2), assuming that clients will select the most popular portion has a substantial probability of error.

In the *prediction* approach, for the particular tree structure considered, $f = 0.35$ results in 73% of the media portions being in the set $\mathcal{F}$, i.e., 73% of the media por-

tions are such that client selections to these portions are always correctly predicted. With this fairly accurate path prediction, the *prediction* approach can achieve reasonably low server bandwidth usage.



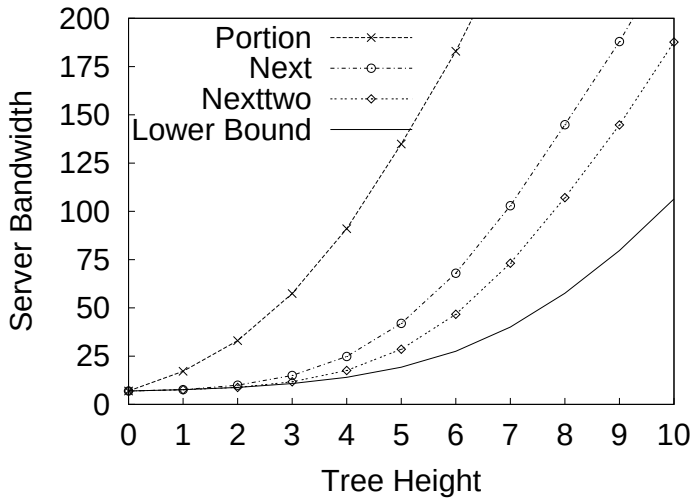

(a) Distance-based Policies



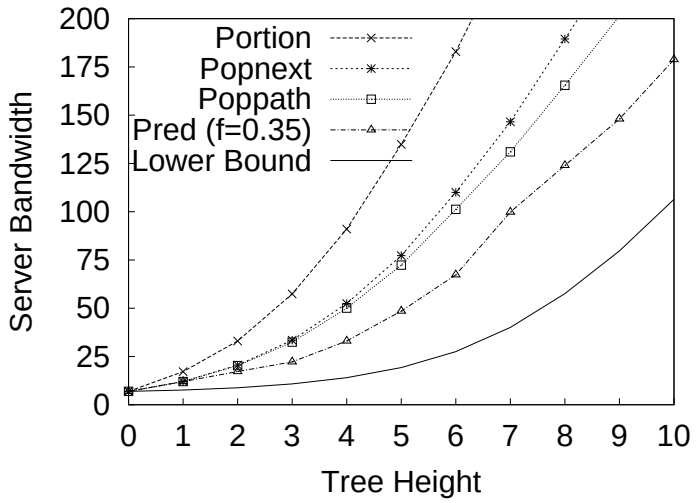

(b) Prediction-based Policies

Figure 5.7: Server Bandwidth for Restricted Snoop-ahead Approaches
(balanced binary tree with height 3, $\alpha = 1$, $d = 0$)

Figure 5.8 shows the impact of the height of the non-linear media tree structure on the server bandwidth requirements of restricted snoop-ahead approaches, assuming immediate service (i.e., $d = 0$), and a fixed client request rate of 1000. Note that for media tree structures of low to moderate height, i.e., height less than four, *next* and *nexttwo* achieve very close to the lower bound server bandwidth requirement.




(a) Distance-based Policies




(b) Prediction-based Policies

Figure 5.8: Impact of Tree Height on Restricted Snoop-ahead Approaches
(balanced binary tree, $\alpha = 1$, $N = 1000$, $d = 0$)

Figure 5.9 presents the minimum client data overhead of various snoop-ahead approaches, as well as of *unrestricted snoop-ahead*. It can be seen from the figure that the *next* and *nexttwo* approaches result in much less client data overhead than the *unrestricted snoop-ahead* approach. The *popular next* and the *popular path* approaches result in even lower client data overhead than *next* and *nexttwo*, but (from Figure 5.7) at the cost of a substantially higher server bandwidth. Compared to these approaches, the *prediction* approach achieves a good balance between server bandwidth reduction and client data overhead.

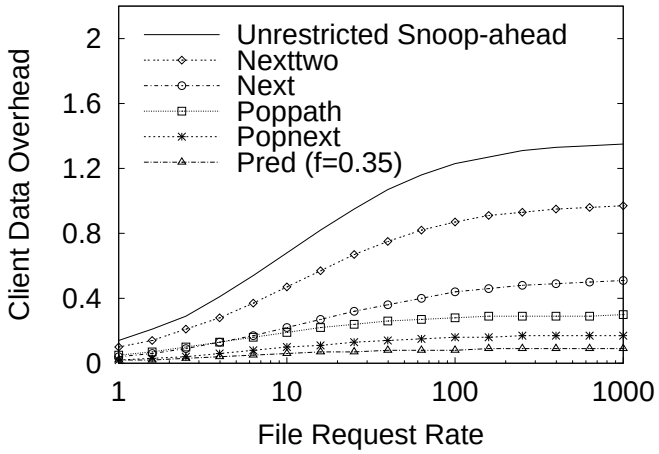



Figure 5.9: Client Overhead with Alternative Snoop-ahead Approaches
(balanced binary tree with height 3, $\alpha = 1$, $d = 0$)

Figure 5.10 graphs the client data overhead of various snoop-ahead approaches as a function of the height of the media tree structure, assuming immediate service (i.e., $d = 0$) and a fixed client request rate of 1000. The results in this figure are consistent with those in Figure 5.9.

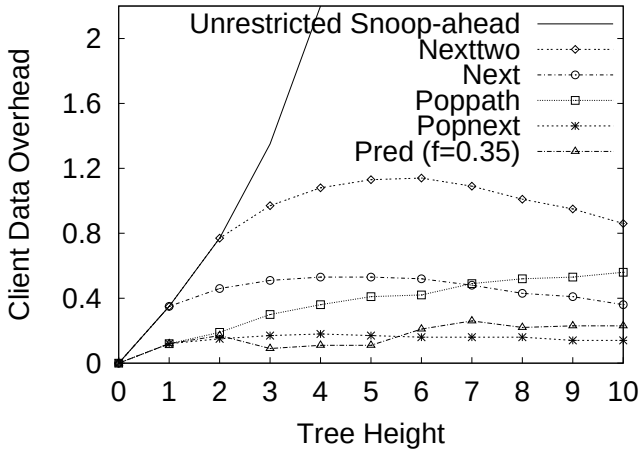



Figure 5.10: Impact of Tree Height on Snoop-ahead Approaches
(balanced binary tree, $\alpha = 1$, $N = 1000$, $d = 0$)

Figure 5.11 shows the sensitivity of the server bandwidth requirement and client data overhead to the skewness in the path selection probabilities, specifically to the Zipf distribution parameter $\alpha$, for various snoop-ahead approaches. For clarity,

results for the *popular next* and the *nexttwo* approaches are not shown, but they are similar to those for *popular path* and *next*, respectively. For the *prediction* approach, the value of the parameter $f$ is varied such that the set $\mathcal{F}$ always contains the same media portions as when $f = 0.35$ and $\alpha = 1$, in which case 73% of media portions are in the set $\mathcal{F}$.

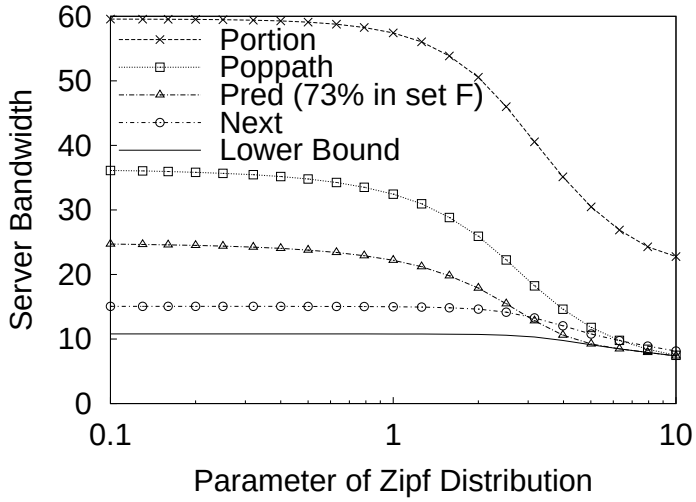
Figure 5.11 shows that the server bandwidth requirement and the client data overhead of the snoop-ahead approaches considered here are quite insensitive to the value of $\alpha$ when $\alpha \leq 1$. As the value of $\alpha$ increases beyond 1, both server bandwidth and client data overhead decrease, since increasingly clients take a common path through the media file. Consistent with the earlier figures, the results suggest that the *next* approach and the *prediction* approach with quite accurate path prediction are promising delivery approaches in that they can achieve low required server bandwidth at a reasonable cost in client data overhead.

## 5.3 Scalable Delivery Protocols

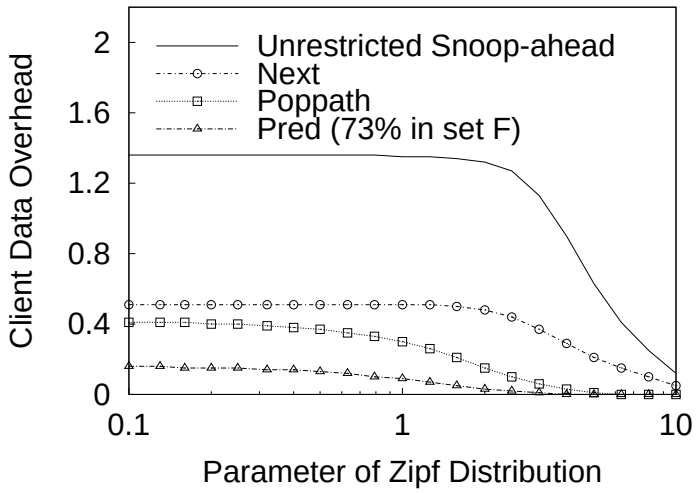
This section addresses the problem of devising efficient scalable delivery protocols for non-linear media. Section 5.3.1 develops variants of hierarchical stream merging (HSM) [36, 38] that provide immediate service delivery of non-linear media. Section 5.3.2 presents non-linear media periodic broadcast protocols based on the optimized periodic broadcast (OPB) protocol [76]. The performance of these protocols with respect to required server bandwidth is evaluated in Section 5.3.3 using simulation, with the lower bound on required server bandwidth given in equation (5.1) as a baseline for comparison.

### 5.3.1 Hierarchical Stream Merging

This section discusses a number of issues that must be addressed if hierarchical streaming merging is to be applied to non-linear media, and candidate strategies for dealing with these issues.




(a) Server Bandwidth




(b) Client Overhead

Figure 5.11: Sensitivity to Skewness in Selection Probabilities
(balanced binary tree with height 3, $N = 1000$, $d = 0$)

First, multicast groups must be organized more dynamically than those for linear media stream merging. With non-linear media, a multicast group may *split* at a branch point when the clients in a group select different following media portions. How the server is able to handle the splitting depends on how much advance knowledge the server has regarding client path selection. At the time that the earliest client in a multicast group (with respect to playback position) makes a selection at a branch point, if the server knows *a priori* that some clients in the same group will choose other portions, the server can choose to split the multicast group into

multiple sub-groups at that time, each consisting of clients that will select (or are expected to select) the same next portion. The group stream will continue beyond the branch point, delivering data from the selected portion to the sub-group containing the earliest client. For each of the other sub-groups, the server may either immediately initiate a new stream for the respective media portion following the branch point, or postpone the initiation of the stream until the earliest client in the sub-group reaches the branch point. On the other hand, if at the time the earliest client in a multicast group reaches a branch point, the server does not know what selections the other clients will make, it may keep the multicast group intact. In this case, all of the clients in the group will continue listening to the group stream as the stream begins to deliver data from the portion selected by the earliest client, in the hope that later clients in the group will choose the same portion. If a client eventually selects a different portion at the branch point, the client will leave the multicast group and the server must initiate a new stream with the data from the respective portion.

Second, it is more complex to determine which previous stream the group should snoop on when all previous streams are delivering data from different paths or have already crossed the next branch point. If the server does not have any advance knowledge of client path selection, one option is simply to instruct the group to snoop on the closest previous stream that is delivering data from the subtree rooted at the current portion. If the server has *a priori* knowledge of client path selection or can accurately predict the path each client will take, it may instruct each client to snoop on the closest previous stream that is delivering a media portion along the path the client will (or is expected to) take, should such a stream exist.

Third, at the time a later client group catches up to a previous group, the clients in the previous group may have already buffered some data through snooping on an earlier stream. These clients could then discard the buffered data because the clients from the later group do not have the data buffered and thus the server may need to deliver it again anyway. A new merge target could then be determined for the entire group. An alternative strategy is that the clients in the previous group could keep

their buffered data, and continue snooping on their current merge target(s). New, possibly distinct, merge target(s) could be determined for the clients from the later group. Note that with this alternative, clients in a single group may be snooping on different streams and have different merge targets, implying that the clients within a single group may accomplish merges at different times and with different prior groups.

A number of non-linear media hierarchical stream merging variants that differ in how the above issues are handled are proposed and described below.

### HSM-UP-G

*HSM Unknown Path Group* (HSM-UP-G) is the variant that most directly applies hierarchical stream merging as devised for linear media to non-linear media. It does not require any *a priori* knowledge of client path selection. Each client (or group of clients) simply snoops on the closest earlier stream that is delivering data from the same portion as the client's stream, if such a stream exists. If there is no such stream, the client or the client group will snoop on the closest earlier stream that is transmitting data from a portion in the sub-tree rooted at the client stream's current portion, should such a stream exist. Note that with the above policy, all clients in a multicast group will have the same merge target. A merge occurs when all of the clients in a later group have received all of the data (by listening to their own stream and snooping on the merge target stream) on their path up to the current portion of the earlier group's stream.

When the earliest client in a multicast group selects a media portion at a branch point, the group's stream will continue past the branch point, delivering data from the selected portion, and all clients in the group will continue listening to that stream. If the group is simultaneously snooping on a previous stream, the server will check whether or not the previous stream is still a valid merge target. If not, the group will be assigned a new merge target, if possible. On the other hand, any other groups that are snooping on the group's stream will continue snooping on this stream since it is still a valid merge target for those other groups.

If a subsequent client in a multicast group later selects a different portion at a branch point than the one selected by the earliest client in the group, the client will leave the current group. The server will initiate a new stream for this client. The client will stop snooping on the group's current merge target stream (if any) and a new merge target will be assigned, if possible. For an existing multicast group that already has a merge target, the new stream does not cause any change in this merge target. For any group without a merge target, the server will check to see if the new stream is a valid merge target.

### HSM-UP-I

*HSM Unknown Path Individual* (HSM-UP-I) operates the same as HSM-UP-G except that each of the clients in a group is able to merge with an earlier group when it has acquired all of the data on its path up to the current portion of the earlier group's stream. Clients need not wait until *all* of the clients in their group have satisfied this condition. When a client merges with an earlier group, it will start to snoop on the earlier group's merge target stream, since the same criterion is used to find the merge target. However, the existing clients from the earlier group will be able to accomplish the merge sooner, since they have already been listening to the merge target stream for some period of time.

### HSM-KP-G

The HSM-UP-G and HSM-UP-I protocols described above assume that the server does not have any *a priori* knowledge of client path selection. In contrast, *HSM Known Path Group* (HSM-KP-G) and HSM-KP-I (described in the next section) assume that the server has full knowledge of client path selection (through perfect client classification, or pre-selection). Given this knowledge, a policy could be developed in which only clients taking exactly the same path can be in the same group. This is analogous to the *path* approach discussed in Section 5.2.1. However, as shown in Figure 5.3, the *path* approach is inefficient in its server bandwidth usage since it does not take advantage of many opportunities for sharing server transmissions. For

example, in the scenario shown in Figure 5.12, the playback paths taken by the two clients $C_1$ and $C_2$ differ only in the leaf portions. Using *path*, the server needs to deliver two separate streams, one for each complete playback path. However, assuming that these two clients made these requests at approximately the same time, the server could serve them more bandwidth-efficiently by using one stream to deliver the data from the beginning to the branch point $B3$, and then two separate streams to deliver portions $D$ and $E$, respectively.

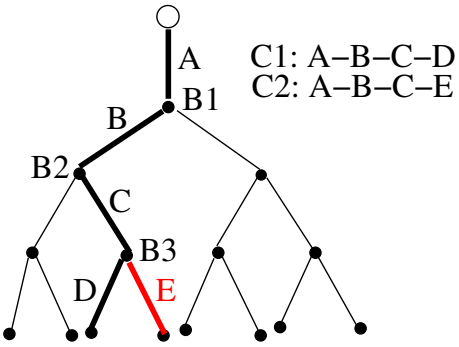



Figure 5.12: An Illustration of the Inefficiency of *Path*

HSM-KP-G and HSM-KP-I are designed to exploit more sharing possibilities than are possible with the *path* approach, while using advance knowledge of client path selection. In HSM-KP-G, the merge target stream of each client is chosen as the closest earlier stream that is currently transmitting data from a portion along the client's path. Note that clients in the same group may have different merge targets. A *merge* occurs when a later client has received all of the data on its path up to the current portion of the merge target stream. Before a merge can take place, the merge target stream may cross a branch point and begin to deliver data from a different portion than what will be selected by the client. In that case, a new merge target stream must be selected. If the merge does happen, the later client will leave its current multicast group, join the merge target group, and begin to snoop on a new merge target stream if possible. In this case, it is likely that the merge target stream will continue for quite some time, as the new client has not yet snooped on any earlier stream and thus its next merge with an earlier stream,

if any, may take some substantial time in the future. Therefore, there may be little benefit to allowing the other clients in the merge target group to merge with earlier streams more quickly and leave the group, because such merges may prolong the life of the earlier streams with which they merge, thus incurring possibly substantial cost. For this reason, the other clients in the group are required to *restart* their merge operations, so that their merges occur no quicker than if they had discarded any previously received snooped data, at the time of the merge by the later client.

When the earliest client in a multicast group selects a portion at a branch point, and not all clients in the group will select this path, the group is split into multiple groups, each composed of clients taking a different branch. The original group's stream continues on past the branch point, delivering data from the portion selected by the earliest client, thus serving the corresponding new group. Depending on when the server begins streams for the other new groups, two variants of HSM-KP-G can be defined. In the first variant, the server immediately begins new streams, starting from the branch point, for the other groups. As a result, the playback positions of all of the clients in each of these groups will be behind the position in the file of the corresponding server stream. In the second variant, the server delays the initiation of a new stream for each group until the playback position of the earliest client in that group reaches the branch point. In this variant, the file position of each stream is the same as the playback position of the earliest client in the corresponding group.

Simulation results indicate that the difference in required server bandwidth of the two variants is negligible. Since the second variant may better handle the case in which *a priori* path selection information is occasionally incorrect, it is the variant used in the performance comparisons shown below.

### HSM-KP-I

*HSM Known Path Individual* (HSM-KP-I) differs from HSM-KP-G in that when two groups of clients merge, the clients in the earlier group are not required to restart their merge operations.

HSM-KP-I also has two variants depending on whether the server immediately initiates new streams for all new groups when a group splits at a branch point, or postpones initiation of each stream (except that for the overall earliest client's group) until the playback position of the earliest client in the corresponding new group reaches the branch point. It was found through simulation that the difference in required server bandwidth of these two variants is negligible, and only results for the variant where the server postpones initiation of new streams are presented here.

Figure 5.13 shows the server bandwidth required by HSM-UP-G, HSM-UP-I, HSM-KP-G, and HSM-KP-I as a function of the normalized file request rate $N$. The figure shows that *a priori* knowledge of client path selection can be used to greatly reduce server bandwidth usage. Since the HSM-UP-G and HSM-KP-G policies have lower required server bandwidth than the HSM-UP-I and HSM-KP-I policies, respectively, in the remainder of this section, HSM-UP-G and HSM-KP-G are used to represent HSM policies without/with advance knowledge of client path selection. HSM-UP-G is referred to as HSM-UP, and HSM-KP-G as HSM-KP.

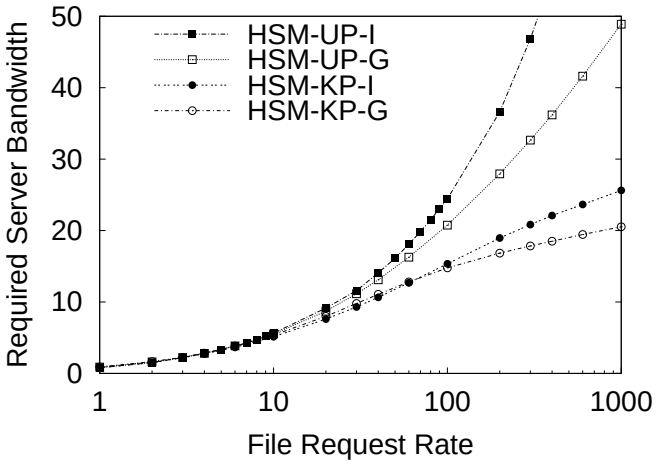



Figure 5.13: Server Bandwidth Requirements of HSM Policies
(balanced binary tree with height 3, $\alpha = 1$)

### HSM-POP

Similar to HSM-UP, the *HSM Popular* (HSM-POP) policy does not require *a priori* knowledge of client path selection. Each client (or group of clients) snoops on the closest earlier stream that is delivering data from the same portion that the client (or client group) is receiving, if such a stream exists. If no such stream exists (i.e., all previous streams that have included this portion have passed the next branch point), the client (or client group) will snoop on the closest stream among those earlier streams that are currently delivering data from the portion with the highest selection probability, among all portions with active streams in the subtree rooted at the portion the client is currently receiving. This choice is motivated by the higher probability of clients taking the corresponding path and thus using the snooped data. The process of group splitting at a branch point and the handling of group merges are as in HSM-UP (i.e., HSM-UP-G). Since Figure 5.13 has shown that *group* policies always perform better than *individual* policies, an *individual* policy variant of HSM-POP is not considered in this section.

Figure 5.14 compares the server bandwidth requirements of *HSM-UP* and *HSM-POP* with varying normalized file request rate $N$. HSM-UP, where client groups simply select the closest previous stream as the merge target stream, has essentially the same required server bandwidth as HSM-POP, where clients snoop on the closest earlier stream that is either transmitting the same portion or on the most popular path if all candidate streams have passed the next branch point. This result is consistent with the result in Section 5.2.2 that using overall path selection probabilities to determine which multicast transmissions to receive may not be a fruitful strategy. Because HSM-POP and HSM-UP have essentially identical performance, in the following figures, the results for HSM-POP are not presented.

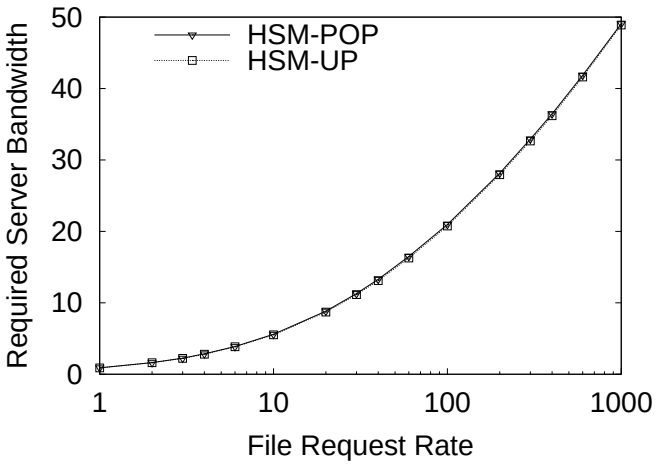



Figure 5.14: Server Bandwidth Requirements of HSM-POP
(balanced binary tree with height 3, $\alpha = 1$)

### HSM-NEXT

HSM-UP and HSM-KP are at two extreme points with respect to the system's knowledge of client path selection. The *HSM Next* (HSM-NEXT) policy occupies an intermediate position in which the system has partial knowledge of client path selection. Specifically, it is assumed that for each active client the system knows the choice that client will make when its playback reaches the next branch point (if any), but not (yet) its choices at subsequent branch points.

Each client selects the closest previous stream that is delivering data from the same portion as the client's stream as the merge target stream, if such a stream exists. If no such stream exists, but there is a stream delivering data from the portion that the client will choose at the next branch point (the system knows the client's choice), the closest such stream will be selected as the merge target stream. If such a stream does not exist, the closest earlier stream that has delivered the current portion and the portion after the next branch point that the client will choose, is selected as the merge target stream, should such a stream exist.

Note that clients in the same group may select different merge targets. HSM-NEXT handles group splitting and group merging similarly as HSM-KP-G, since as illustrated in Figure 5.13 *group* policies are more efficient than *individual* policies.

Figure 5.15 presents the server bandwidth requirements of HSM-UP, HSM-NEXT, and HSM-KP, as a function of the normalized file request rate $N$. The figure illustrates that partial knowledge of client path selection can be used to make better choices of merge targets, thus reducing server bandwidth usage.

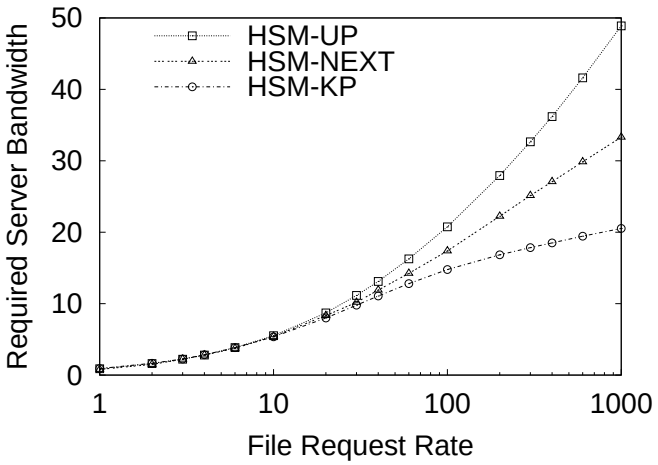



Figure 5.15: Server Bandwidth Requirements of HSM-NEXT
(balanced binary tree with height 3, $\alpha = 1$)

### 5.3.2 Optimized Periodic Broadcast

In this section, periodic broadcast protocols for non-linear media are developed. Specifically, these protocols are based on the optimized periodic broadcast (OPB) protocol [76]. Using the notation summarized in Table 5.2, in the OPB protocol, a file is partitioned into $M$ segments, each repeatedly broadcast on a separate channel at rate $r$. The segment size progression is chosen such that if clients begin listening to the $s$ channels delivering the first $s$ segments immediately after tuning in, switch from channel $m$ to channel $m + s$ $(m + s \leq M)$ after segment $m$ is fully received, and begin playback after the first segment is received in its entirety, every segment can be received just in time for playback.

In this section, two variants of the OPB protocol are designed: *OPB Known Path* (OPB-KP) (in Section 5.3.2.1) for the case in which client path selection is known *a priori*, and *OPB Unknown Path* (OPB-UP) (in Section 5.3.2.2) for the case where client path selection decisions are known only when they are made at the respective branch points.

Table 5.2: Notation for OPB

| Symbol | Definition |
|---|---|
| $M$ | Number of segments |
| $r$ | segment transmission rate (in units of the file playback bit rate) |
| $s$ | Maximum number of streams that clients listen to concurrently ($s \leq M$) |
| $b$ | Maximum aggregate sustainable reception rate of a client (in units of the file playback bit rate), $b = sr$ |
| $l_m$ | Playback duration of segment $m$ |

#### 5.3.2.1 OPB Known Path (OPB-KP)

OPB-KP is designed under the assumption that the system has full knowledge of client path selection. This knowledge could be obtained by clients pre-selecting playback paths or by accurately classifying clients based on their past behaviours. In OPB-KP, each complete path of a non-linear media file is partitioned using the same segment size progression as in OPB for linear media files, but with a given *absolute* rather than relative length for the first segment. Longer paths will end up with more segments, and the last segment on a path may end up being a partial segment. For a non-linear media file structured as an arbitrary tree, shared portions on paths share the corresponding segments. For a non-linear media file with a directed acyclic graph structure, the same property holds if the path lengths to any media portion with multiple parents are identical.

If a segment crosses over a branch point, the data before the branch point is still transmitted on a channel at rate $r$, as is done in OPB for linear media, but the

data on each media portion after the branch point is delivered on a separate *sub-channel*, each at rate $r$. Figure 5.16 illustrates how the server broadcasts segment $m$ that crosses over branch point $B$. Repeatedly, the server first transmits data $X$–$B$ (at rate $r$), and then data $B$–$Y$ and $B$–$Z$ (at a total rate of $r$ times the number of portions after the branch point). Each client listens to the channels and sub-channels corresponding to its path. If a OPB/tou-like policy, as described in Section 3.5, is used, that is, the server can determine if there are listeners on a channel (or sub-channel) and stop transmission on the channel if there is not, this scheme is efficient with respect to bandwidth usage.

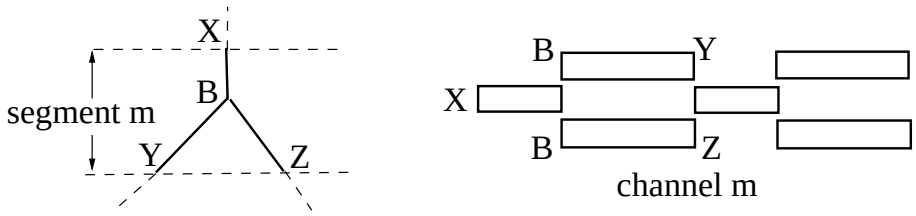



Figure 5.16: Broadcasting a Segment that Crosses a Branch Point

Figure 5.17 shows the partition of an example non-linear media file into three segments (i.e., $M = 3$), assuming that clients listen to two channels concurrently (i.e., $s = 2$) and that segments are broadcast at the file playback bit rate (i.e., $r = 1$). Figure 5.18 illustrates how these segments are broadcast on channels and sub-channels. The shaded areas in Figure 5.18 are listening periods for the example client, assuming that the client request arrives as marked and that the client takes the path $S - B1 - B2 - E5$ on the non-linear media tree shown in Figure 5.17.

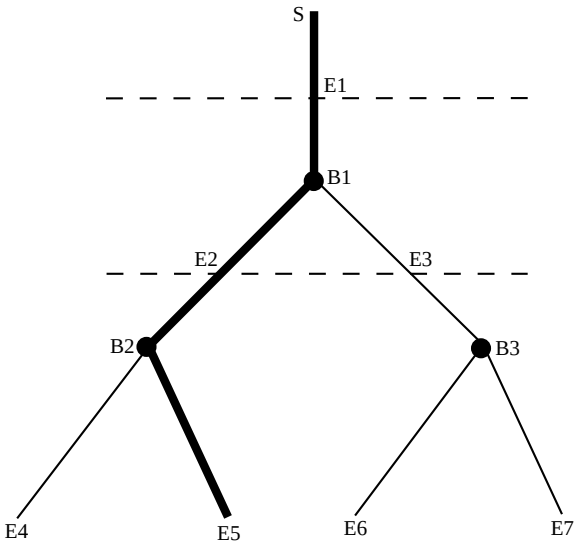



Figure 5.17: OPB-KP Segment Partitioning for an Example Media Structure
("S" label is for start of media, "Bi" labels are for branch points, "Ei" labels are for segment end points, dashed lines indicate segment boundaries, $M = 3$, $s = 2$, $r = 1$)

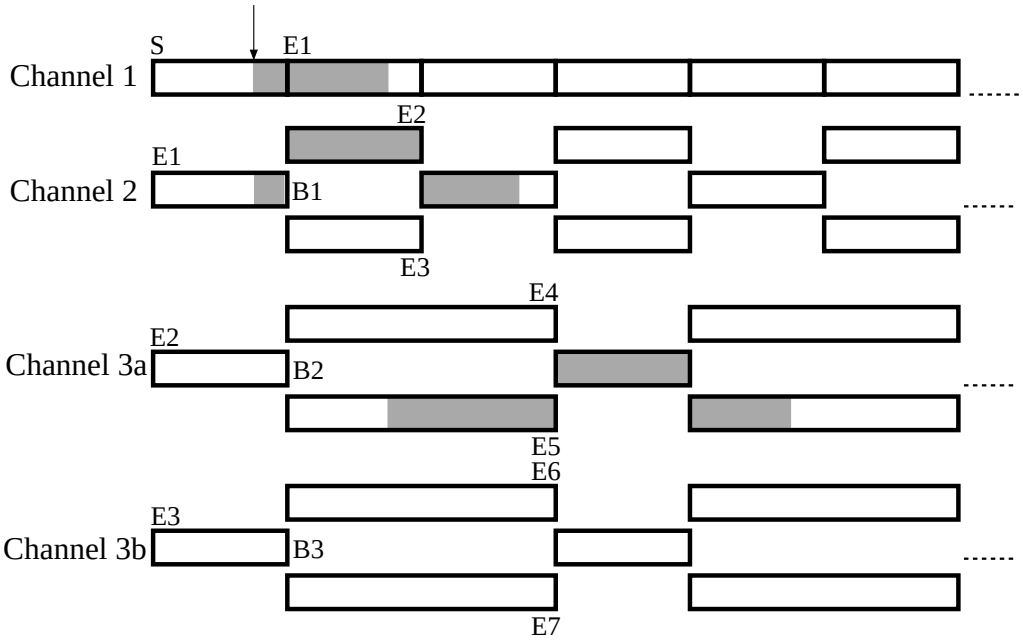



Figure 5.18: OPB-KP Channels for Structure of Fig. 5.17
(shaded areas are listening periods of example client, $M = 3$, $s = 2$, $r = 1$)

The server bandwidth requirement of OPB-KP, assuming that the server transmits on a channel (or sub-channel) only when there is at least one listener, can be computed by taking the sum of the required server bandwidth for broadcasting each segment. For a segment $m$ that does not cross a branch point (e.g., the first segment in Figure 5.17), the required server bandwidth, assuming Poisson arrivals, can be computed as:

$$r\left(1-e^{-\lambda_i \frac{l_m}{r}}\right), \tag{5.14}$$

where $1-e^{-\lambda_i \frac{l_m}{r}}$ is the probability that there is at least one client listening to the channel delivering segment $m$.

For a segment $m$ that extends over a branch point, e.g., the segment shown in Figure 5.19, the server bandwidth requirement can be computed as:

$$r\left(1-e^{-\lambda_j \frac{l_m}{r}}\right)+r\left(1-e^{-\lambda_k \frac{l_m}{r}}\right)-r\frac{l_a}{l_m}\left(1-e^{-\lambda_j \frac{l_m}{r}}\right)\left(1-e^{-\lambda_k \frac{l_m}{r}}\right), \tag{5.15}$$

where the sum of the first two terms is the required server bandwidth for transmitting data $X$–$Y$ and $X$–$Z$ on two separate channels. Since data $X$–$B$ is transmitted only on one channel, the extra bandwidth, computed by the third term, is subtracted from the total. In the third term, $(1-e^{-\lambda_j \frac{l_m}{r}})(1-e^{-\lambda_k \frac{l_m}{r}})$ is the probability that at least one client listening to the channel delivering data $X$–$B$, and $r\frac{l_a}{l_m}$ is the bandwidth required to transmit data $X$–$B$.

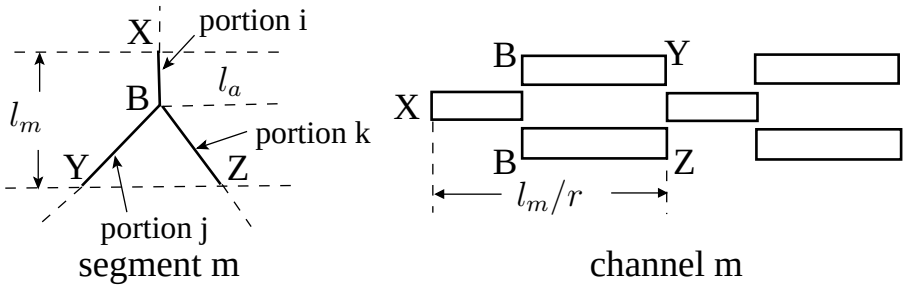



Figure 5.19: Server Bandwidth Usage for a Cross-over Segment

Note that OPB-KP represents the best case with respect to the required server bandwidth of the OPB protocol for non-linear media delivery. If client path selection cannot be perfectly predicted by the system, which is likely the case in practice, recovering from wrong path predication would be difficult, which may result in interruptions in playback, or the delivery of unicast streams to recover the unpredicted data required by clients.

#### 5.3.2.2 OPB Unknown Path (OPB-UP)

OPB-UP is designed for the case where client path selection decisions are known only when they are made at respective branch points. Here it is assumed a segment never crosses more than one branch point. In OPB-UP, for a segment $m$ that crosses a branch point $B$, the data from branches after the branch point can no longer be broadcast in sub-channels, as was done in OPB-KP, since a client cannot know which sub-channel to listen to when it starts to download segment $m$. Instead, the data from all branches after branch point $B$ are *multiplexed* and transmitted on one channel. The rationale is that the client can be guaranteed to obtain some useful data for playback after it makes its decision at the branch point. How segment data prior to the branch point is transmitted depends on whether or not the file playback crosses a branch point $B'$ between a client begins to listen to the segment $m$ and when the playback of that segment begins. If the file playback does not cross a branch point, the data of segment $m$ prior to branch point $B$ must be from the same media portion as that being played back during the reception of segment $m$. In that case, this part of segment $m$ is broadcast on a channel. On the other hand, if the file playback does cross branch point $B'$, the data of segment $m$ prior to branch point $B$ must be *multiplexed* because segment $m$ must begin after branch point $B'$, and it is unknown which branch a client will take at branch point $B'$ when the reception of segment $m$ begins.

Moreover, if segment $m$ does not cross a branch point $B$, and if the file playback also does not cross a branch point between when a client begins to listen to segment $m$ and when the playback of that segment begins, segment $m$ must be part of the

same media portion that was being played back during its reception. Segment $m$ will be broadcast on a separate channel. On the other hand, if the file playback does cross a branch point $B'$, segment $m$ must contain data from media portions after branch point $B'$. These data must be *multiplexed* since at the time a client starts to receive segment $m$, the client does not know which portion to take after branch point $B'$.

Figure 5.20 shows the partition of a non-linear media file into six segments (i.e., $M = 6$) using OPB-UP, assuming that clients listen to two channels concurrently (i.e., $s = 2$), and that segments are transmitted at the file playback bit rate (i.e., $r = 1$). Note that the second segment crosses the first branch point of the non-linear media file. Figure 5.21 illustrates how these segments are broadcast on channels. The stripes indicate that the data are multiplexed. The shaded areas represent the periods during which an example client, which makes the request for the file at the time marked by the arrow and takes the path highlighted in Figure 5.20, listens to the transmission on each channel.

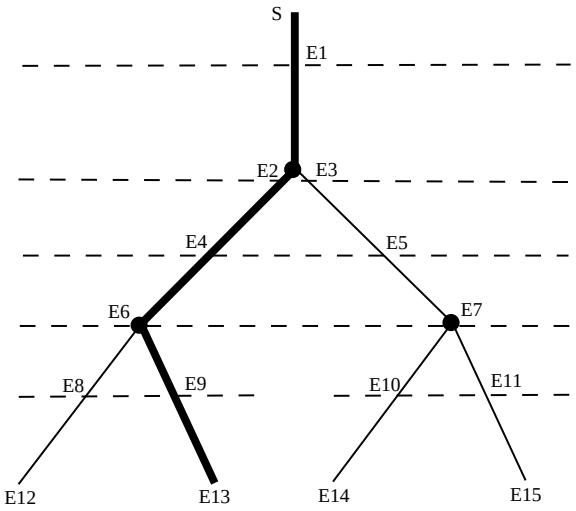



Figure 5.20: OPB-UP Segment Partitioning for an Example Media Structure
("S" label is for start of media, "Bi" labels are for branch points, "Ei" labels are for segment end points, dashed lines indicate segment boundaries, $K = 3$, $s = 2$, $r = 1$)

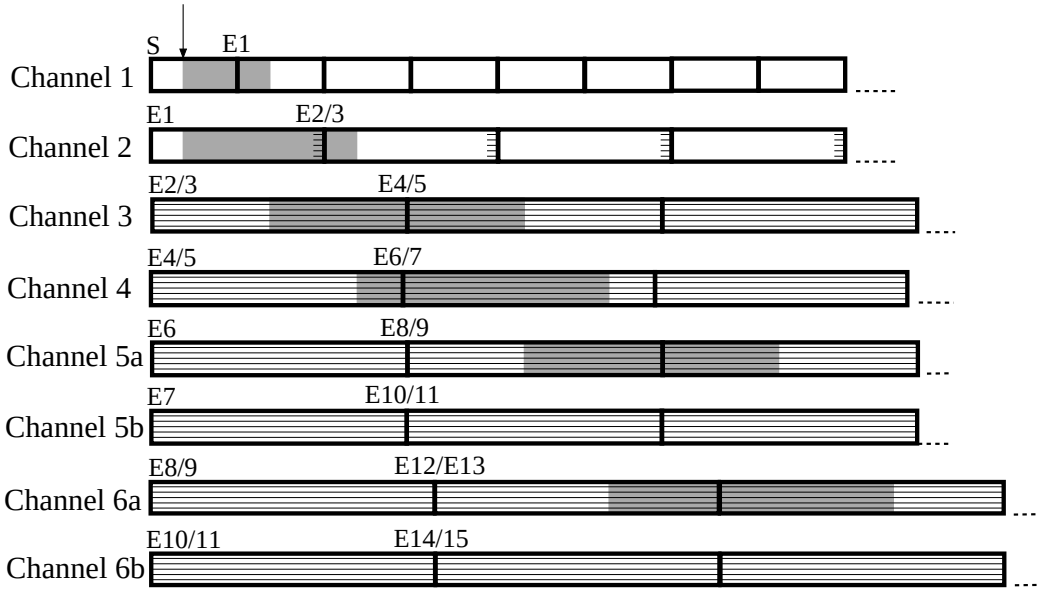



Figure 5.21: OPB-UP Channels for Structure of Fig. 5.20
(shaded areas are listening periods of example client, $K = 3$, $s = 2$, $r = 1$)

OPB-UP cannot use the same segment size progression as in OPB for linear media, since the playback duration of a segment that contains multiplexed data is less than what the segment size in bytes (and corresponding transmission time) would suggest, because a client will playback only the data on its chosen path. Figure 5.22 outlines the algorithm used in this study to compute a feasible segment size progression for OPB-UP. The notation employed in the algorithm is summarized in Table 5.3. Although this algorithm is designed for balanced binary trees, it can be extended for arbitrary tree structures. In the main function *Partition*, the outer loop attempts to find a start-up delay (i.e., transmission time of the first segment) such that the cumulative length of $M$ segments (where $M$ is given as a input) matches the length of a complete playback path. To simplify the design, the algorithm makes the following assumptions: a) a segment never spans over more than one branch point; b) the first segment does not cross a branch point; c) branch points are never sufficiently close together that a zero length is computed for a segment (as would happen in Case 2.2 when branch point $B$ is at $e_k$)[1]; and d) a client begins the

[1]this case can be handled by delaying the time a client starts to receive the segment until the client makes its choice at the branch point.

reception of a new segment immediately after the reception of a previous segment completes. Note that in Case 1.3 and Case 2.2, segment $m$ ends at a branch point even though it could be longer according to the segment size progression if it did not encounter the branch point. This special handling is added to avoid a segment having a multiplexing level of more than two levels of media portions.

Table 5.3: Notation for OPB-UP

| Symbol | Definition |
|---|---|
| $M$ | Number of segments |
| $d$ | Maximum client start-up delay |
| $D$ | Client start-up delay expressed as a fraction of the file playback duration ($D = d/T$) |
| $l_m$ | Playback duration of segment $m$ |
| $u_m$ | Time when a client begins the reception of segment $m$, relative to the start of file playback |
| $e_m$ | Latest time when a client can end reception of segment $m$, also the starting playback time of segment $m$ |
| $y_m$ | Transmission time of segment $m$ when it is of maximal length |
| $w_m$ | Ending playback time of segment $m$ if the segment does not encounter a branch point |

The server bandwidth requirement of OPB-UP, assuming that the server transmits data on a channel only when there is at least one listener, can be computed by taking the sum of the required server bandwidth for delivering each segment. For a segment $m$, the required server bandwidth is given by

$$r\left(1 - e^{-\lambda_i \frac{l_m}{r}}\right), \tag{5.16}$$

where if segment $m$ does not cross a branch point, $\lambda_i$ is the client request rate to the media portion covered by segment $m$; and if segment $m$ does cross a branch point, $\lambda_i$ is the client request rate to the media portion covered by segment $m$ prior to the branch point. Assuming Poisson arrivals, $1 - e^{-\lambda_i \frac{l_m}{r}}$ is the probability that there is at least one client listening to the channel transmitting segment $m$.

Function $Partition(M, s, r)$
1. For $(i = 0;\ i \leq W;\ i{+}{+})$
2. $\quad d = \epsilon_1 + i(T/r - \epsilon_1)/W$
3. $\quad Compute\_Segsize(M, d, s, r)$
4. $\quad$ If $(\left|\sum_{m=1}^{M} l_m - T\right| \leq \epsilon_2)$
5. $\quad\quad$ Return $Success$
6. End For
7. Return $Failed$
End Function

Procedure $Compute\_Segsize(M, d, s, r)$
8. $l_1 = dr$
9. For $(m = 2;\ m \leq M;\ m{+}{+})$
10. $\quad u_m = -d,\ \ m \leq s;$
$\quad\quad u_m = s$'th latest of $\{\ u_j + l_j/r \mid 1 \leq j < m\ \},\quad m \geq s+1$
11. $\quad e_m = \sum_{j=1}^{m-1} l_j;\ y_m = e_m - u_m$
12. $\quad$ Case 1: no branch point in $(u_m, e_m)$
13. $\quad\quad w_m = y_m r + e_m$
14. $\quad\quad$ Case 1.1: no branch point in $[e_m, w_m)$
15. $\quad\quad\quad l_m = y_m r$
16. $\quad\quad$ Case 1.2: first branch point in $[e_m, w_m)$ is at $B$ and no branch point in $(B, B + (y_m r - (B - e_m))/2)$
17. $\quad\quad\quad$ transmit interleaved data after branch point
18. $\quad\quad\quad l_m = B - e_m + (y_m r - (B - e_m))/2$
19. $\quad\quad$ Case 1.3: first branch point in $[e_m, w_m)$ is at $B$, and first branch point in $(B, B + (y_m r - (B - e_m))/2)$ is at $B_2$
20. $\quad\quad\quad$ segment ends at branch point $B_2$
21. $\quad\quad\quad l_m = B_2 - e_m$
22. $\quad$ Case 2: one branch point in $(u_m, e_m)$
23. $\quad\quad$ transmit interleaved data
24. $\quad\quad w_m = y_m r/2 + e_m$
25. $\quad\quad$ Case 2.1: no branch point in $[e_m, w_m)$
26. $\quad\quad\quad l_m = y_m r/2$
27. $\quad\quad$ Case 2.2: first branch point in $[e_m, w_m)$ is at $B$
28. $\quad\quad\quad$ segment ends at branch point
29. $\quad\quad\quad l_m = B - e_m$
30. End For
End Procedure

Figure 5.22: Algorithm for OPB-UP Segment Sizes
(balanced binary tree)

### 5.3.3 Performance Comparisons

Figures 5.23, 5.24, and 5.25 compare the server bandwidth requirements of HSM (with/without client path prediction) and OPB (with/without client path predication), and the lower bound of expression (5.1), as a function of the normalized file request rate (Figure 5.23), non-linear media tree height (Figure 5.24), and the client start-up delay (Figure 5.25), respectively. It is not surprising to see from these figures that knowing client path selection in advance (i.e., HSM-KP and OPB-KP) substantially reduces the server bandwidth usage in comparison to not having any knowledge of client path selection (i.e., HSM-UP and OPB-UP). Moreover, letting the server shut off a channel whenever there are no clients listening to that channel makes OPB protocols perform comparably to HSM protocols even under light client load (i.e., $N < 10$). Without this optimization, OPB would perform considerably worse than HSM when the client request rate is low. Furthermore, OPB with perfect client path predication (OPB-KP) can achieve server bandwidth close to the lower bound, even though OPB-KP assumes a constrained client receive bandwidth of twice the file playback bit rate, while the lower bound assumes unlimited client receive bandwidth. Note that since those scalable delivery protocols achieve the bandwidth efficiency at a reasonably low cost client data overhead, since these protocols assume a fixed client receive bandwidth.

## 5.4 Prototype Implementation

This section describes the implementation of a simple non-linear media hierarchical stream merging variant in the SWORD (Scalable Wide-area On-demand Reliable Digital Streaming) prototype multimedia streaming system. The SWORD prototype system [109] was developed at the University of Wisconsin-Madison to experiment with scalable delivery protocols for multimedia streaming. Currently, a variant of the hierarchical stream merging protocol for linear media has been implemented in the SWORD prototype, and is being used to deliver on-line courses at the University of Wisconsin-Madison [114].

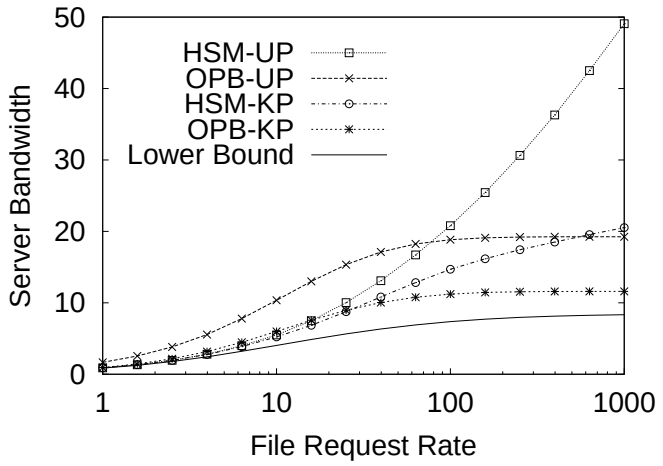



Figure 5.23: Performance of Scalable Delivery Protocols
(balanced binary tree with height 3, $\alpha = 1$, $D = 0.01$ for OPB protocols and the lower bound, $r = 0.25$, $s = 8$)

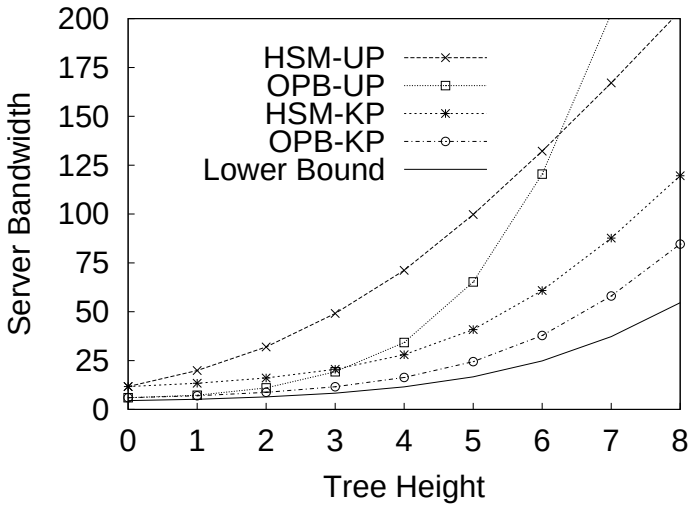



Figure 5.24: Performance with Varying Height
(balanced binary tree, $\alpha = 1$, $N = 1000$, $D = 0.01$ for OPB protocols and the lower bound, $r = 0.25$, $s = 8$)

### 5.4.1 The SWORD Prototype

The SWORD streaming system operates as middleware between a media server and media clients. It adds scalable streaming functions on top of a conventional media streaming service (i.e., the media server delivers a separate stream for each client request). The SWORD system consists of a server component and a client

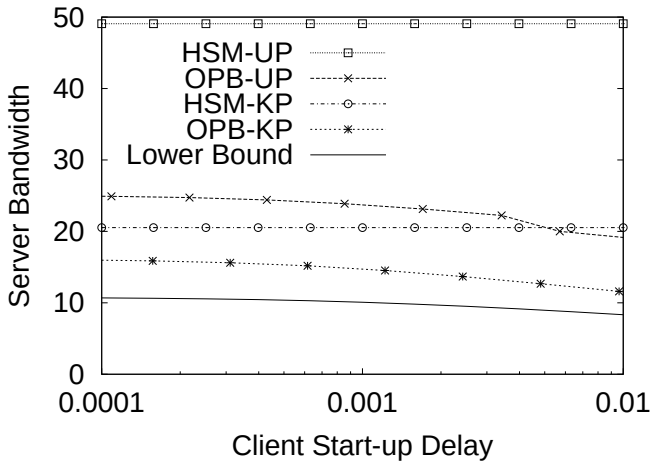



Figure 5.25: Impact of Start-up Delay
(balanced binary tree with height 3, $\alpha = 1$, $N = 1000$, $r = 0.25$, $s = 8$)

component, both of which are *web proxies* modified to use the WWSTP[2] transport protocol for scalable media streaming. Figure 5.26 illustrates how the SWORD server and client components typically interact with the media server and clients. An instance of the SWORD client component is initiated for each media client, and an instance of the SWORD server component is initiated to control all SWORD clients. The SWORD server and clients *transparently* intercept every HTTP communication between the media server and media clients. They forward HTTP requests and HTTP response headers. When a SWORD server receives the HTTP response payload that contains the file data (a TCP stream), it *multicasts* the data using UDP multicast. The SWORD client listens to the appropriate UDP multicast channels as dictated by the SWORD server, and then transmits the data via a TCP stream to the corresponding media client. Under this implementation, the media server still initiates a separate stream in response to each client request it receives, and the media client still receives a stream containing the requested data. However, clients that make closely-spaced requests for the same file can share one multicast stream from the SWORD server to the SWORD clients, and after the clients merge into one group, the SWORD server can explicitly terminate the corresponding TCP streams

[2]WWSTP stands for Wisconsin Washington Saskatchewan Transport Protocol to reflect the names of the universities of researchers that have been involved in the development.

from the media server. Therefore, the server bandwidth usage at the media server and the network bandwidth usage from the media server to the SWORD server and from the SWORD server to the SWORD clients can be reduced, compared to those required in the original media streaming system.

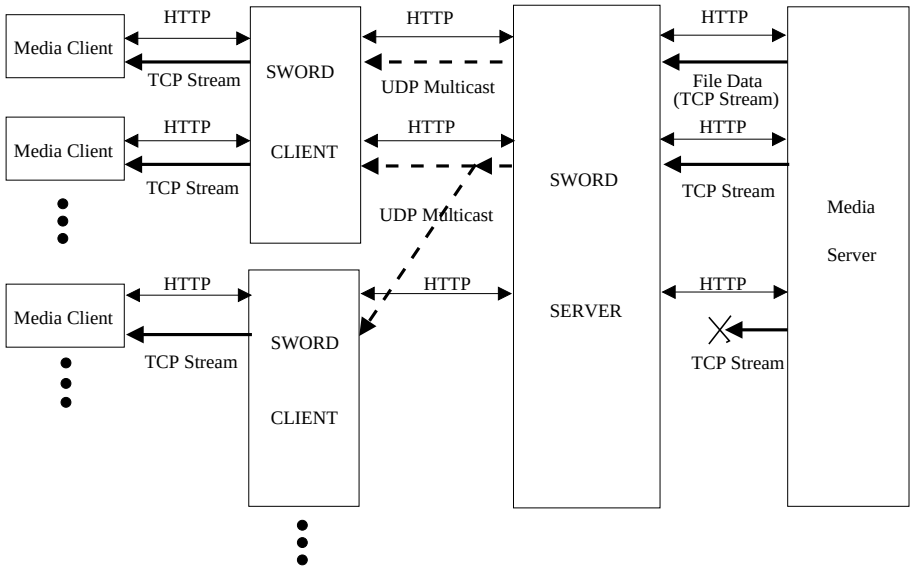



Figure 5.26: SWORD Prototype Architecture

Figure 5.27 illustrates how a multicast is initiated between the SWORD server and the SWORD client. After the SWORD server receives the HTTP response header, it appends a "use WWSTP" field to the header, forwards it to the SWORD client, and then enters the *hand-shaking* process. When the SWORD client receives the modified header, it peels off the "use WWSTP" field, forwards the original header to the media client, and then enters the hand-shaking process. In the hand-shaking process, the SWORD server first creates a new UDP multicast group, and then sends a NEW_CLIENT message with the multicast group address via the TCP connection to the SWORD client. The SWORD client joins the multicast group, starts to listen on that multicast channel, and then sends a JOINING message via the TCP connection back to the SWORD server. Since there are some overheads associated with joining a multicast group, to guarantee complete delivery of file data to the SWORD client, the SWORD server performs a simple test after receiving

the JOINING message by sending PING messages on the UDP multicast channel. On receiving each PING message, the SWORD client replies with a LISTENING message via the TCP connection. After receiving two[3] LISTENING messages, the SWORD server starts to multicast the file data to the SWORD client. The SWORD client forwards the data received on the UDP multicast channel to the media client via the TCP connection. Note that before the hand-shaking process finishes, the media server may have already started transmitting data to the SWORD server. The SWORD server buffers this data until it is able to multicast the data to the SWORD client.

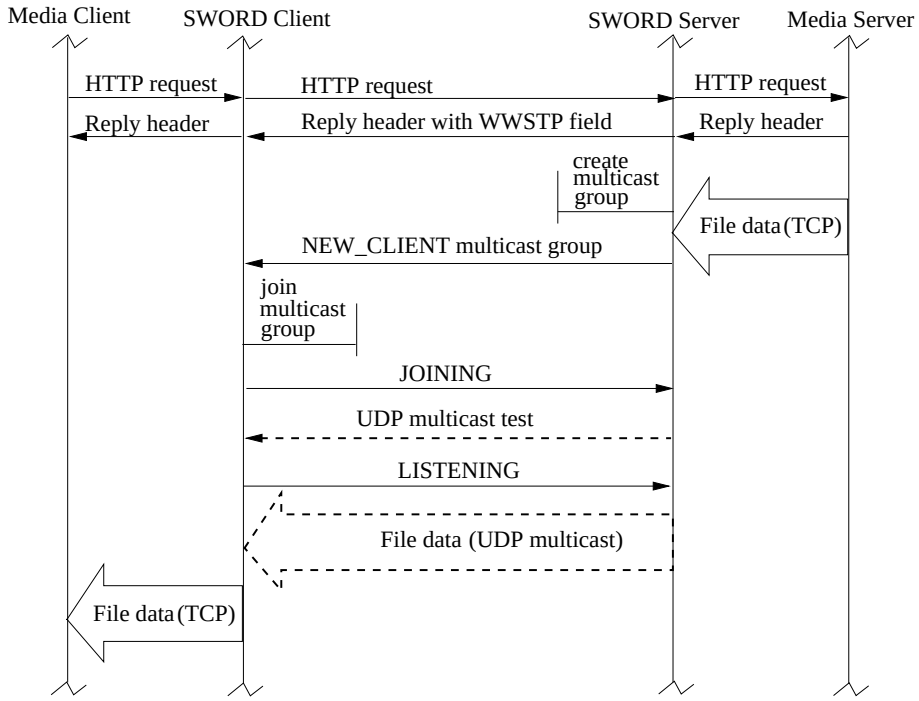



Figure 5.27: SWORD Server/Client Hand-shaking Interaction

Figure 5.28 illustrates how a simple variant of hierarchical stream merging (HSM) is implemented. The SWORD server runs the HSM algorithm at regular intervals. In each round, for each file being delivered, the SWORD server checks all server streams that are transmitting this file. If it finds out that the client group that is listening to such a stream has not snooped on any previous stream, it will find a

[3]a parameter that can be specified

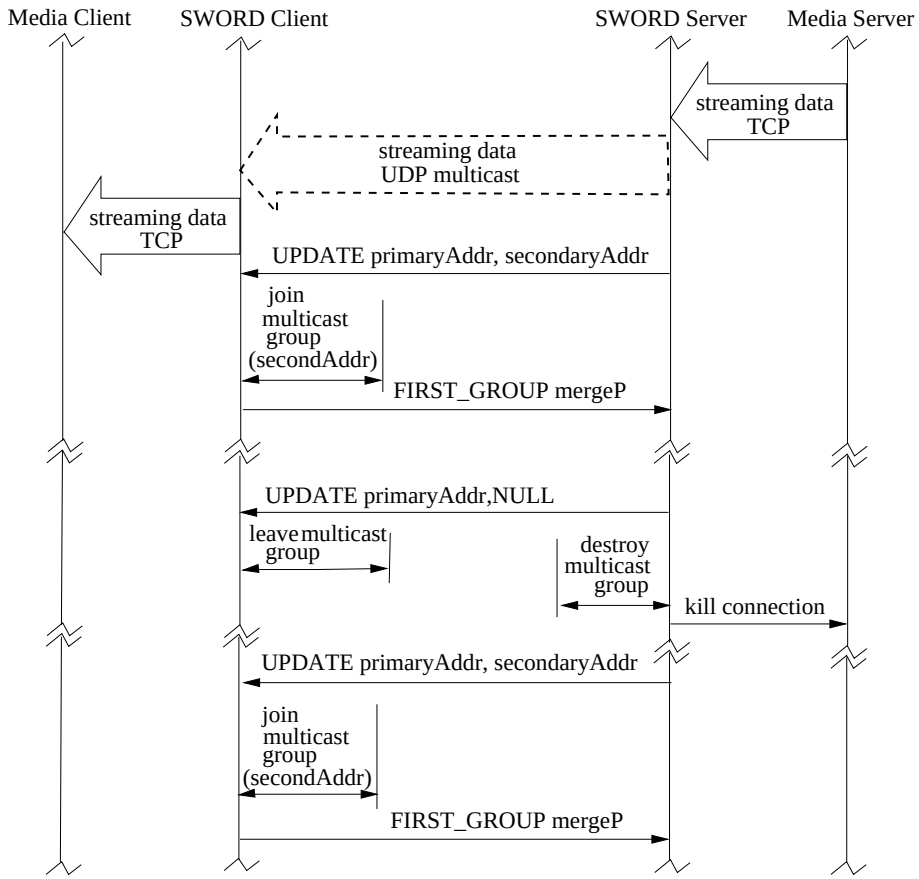



Figure 5.28: HSM in SWORD

merge target stream for this group. Then, the SWORD server sends out an UPDATE message to each client in the group through the corresponding TCP connection. The UPDATE message contains two fields, the multicast address of the stream (i.e., *primary stream*) the client is listening to, and the multicast address of the merge target stream (i.e., *secondary stream*) that the client will snoop on. When a SWORD client receives this message, it will join the multicast group of the merge target stream and start to receive data from that stream. Since usually there are delays between when the SWORD server sends out the UPDATE message and when the SWORD client actually receives data packets from the merge target stream, the SWORD server does not assume that the SWORD client can start to receive the merge target stream at a particular time. Rather, after the SWORD client receives the first packet from the merge target stream, it replies with a FIRST_GROUP message to

the SWORD server, with the packet number of the first packet ($mergeP$) it receives. After receiving the FIRST_GROUP message, the SWORD server remembers that the primary stream the client is currently listening to should be terminated when it is about to transmit the packet $mergeP$, since the client has obtained all the following packets, including the packet $mergeP$, from the merge target stream. If more than one client is currently listening to the primary stream, the SWORD server will receive a FIRST_GROUP message from each client. In this case, the server will record the largest packet number in all FIRST_GROUP messages, since this number marks the point when the latest client in the group catches up to the merge target, and so the primary stream these clients are listening to can be terminated.

When the SWORD server determines that the client group that is listening to a stream has caught up to the merge target stream, the SWORD server sends an UPDATE message to each client in the group. In that message, the SWORD server sets the primary multicast stream address to that of the merge target stream, and the secondary multicast stream address to *NULL*. After receiving this message, the SWORD client stops listening to the original primary stream, and leaves the corresponding multicast group. If the SWORD server finds out that no clients are listening to a server stream, either as the primary stream or the secondary stream, the SWORD server will shut down the corresponding TCP connection from the media server. By shutting off server streams that are no longer needed, the required server bandwidth at the media server and the required network bandwidth from the media server to the SWORD server for delivering these streams are saved.

The bottom part of Figure 5.28 illustrates that when the SWORD server runs a new round of the HSM algorithm, it determines a new merge target stream for the SWORD client and sends out a new UPDATE message to let the SWORD client snoop on that stream.

In the current implementation of SWORD, the SWORD server and client components are developed on top of Apache web proxies. Currently, the media server has to be a Windows Media Server system, but the media client can use Windows Media Player or a Linux media player.

### 5.4.2 Non-linear Media Streaming in SWORD

#### 5.4.2.1 Constructing Non-linear Media Files

Since real non-linear media files have not previously been constructed, the first problem in non-linear media streaming is to obtain non-linear media files. It might be possible to create a non-linear media file using existing multimedia authoring software, e.g., Adobe Premier, although this would not appear to be straightforward. In this study, a non-linear media file is constructed from normal linear media files, with each linear media file representing a portion of the non-linear file. Figure 5.29 illustrates an example of a tree-structured non-linear media file made up of normal linear media files. At the media server, these linear media files are independently stored. Their relations in the non-linear media structure are unknown to the media server.

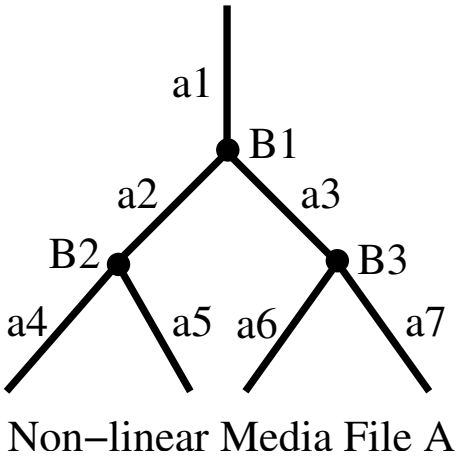



Figure 5.29: An Example Non-linear Media File

Since current media servers and media clients are designed for streaming normal linear media files, the second problem is to make non-linear media streaming transparent to both media server and media client. For media clients, the transparency is realized by assigning each non-linear media file a unique name that is constructed in the same way as for a linear media file, such that the media client can request a non-linear media file in the same way as requesting a linear media file. For example, as shown in Figure 5.30, the media client sends an HTTP request for non-linear

media file $A$, as if it were a normal linear media file, to the media server. After it receives data packets from the SWORD client, it starts to render them to the end-user. The SWORD client, after receiving the request for file $A$, replaces it with the request for linear file $a_1$, which is the first portion of the non-linear media file $A$. The request is forwarded by the SWORD server to the media server, where a file named $a_1$ but not $A$ is stored. After receiving data packets from the SWORD server, the SWORD client makes necessary modifications (which will be discussed in Section 5.4.2.2) so as to make the media client believe that this data is for file $A$.

As shown in Figure 5.30, an end-user can select the next portion at a branch point through a special user-interface. The selection is then sent directly to the corresponding SWORD client. When receiving the selection, e.g., for portion $a_2$ at the branch point $B_1$, the SWORD client will spoof an HTTP request for file $a_2$ and send the request to the media server. After it receives data packets of file $a_2$, it makes necessary modifications such that the media client considers these data packets as follow-up data packets of file $A$.

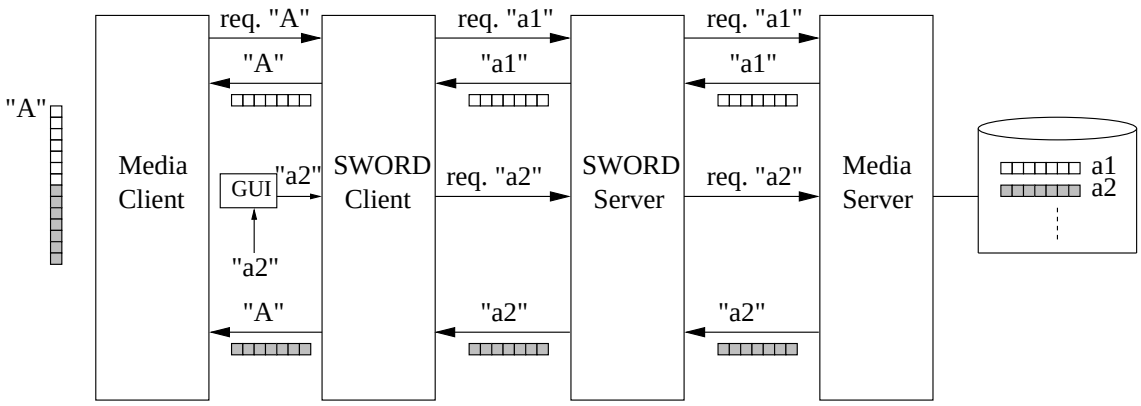



Figure 5.30: Transparency to Media Server and Media Client

Currently, the user-interface has not been developed. The playback paths selected by end-users are hard-coded in the SWORD client component. However, this simplification does not affect the main logic of the SWORD client component.

#### 5.4.2.2 SWORD Client Component Design

As discussed in the previous section, the SWORD client needs to modify both HTTP requests and responses between media client and media server to enable transparent non-linear media streaming. To meet this requirement, the SWORD client must have full knowledge of any requested non-linear media file, so that it knows whether the requested file is a non-linear media file and how to spoof the HTTP requests to the media server if it is. The structure of a non-linear media file can be described using XML-like tags in a text file, and read in by the SWORD client during the initiation stage. Currently, this has not been implemented. All non-linear file structures are hard-coded in the SWORD client component.

It is fairly easy for the SWORD client to modify the HTTP requests and response headers, and to spoof HTTP requests for the follow-up portions at a branch point. However, making the media client believe that multiple linear media files are pieces of a single linear file is a challenge. Currently, Windows Media files with extensions *.wmv* and *.asf*, which follow the Microsoft Advanced System Format (ASF), are able to be "converged" (i.e., pieced together during playback) in the SWORD system, as discussed in the remainder of this section.

A typical ASF-formatted Windows Media file, specifically a movie file, is composed of three high level components: a header object, a data object, and an index object. As shown in Figure 5.31, the *header object* contains general information about the file, including the number of streams contained in the movie (usually only two streams, one audio stream and one video stream), codecs used, file size (in bytes), the number of data packets, data packet size (in bytes), playback duration (in units of 100-nanosecond), etc.. The *data object* contains a header followed by the data packets. The *index object* contains information that is not relevant to the convergence of movie files, but rather is used as an indication of the end of the data object, i.e., the end of all data packets.

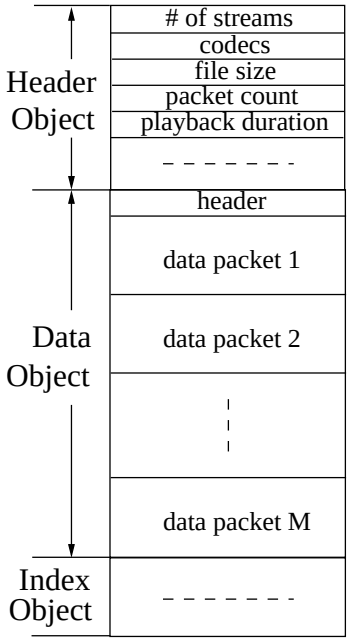



Figure 5.31: An ASF Movie File

A network packet sent out by the media server (specifically, the Windows Media Server) generally corresponds to a data packet, except for the first network packet, which usually contains the header object followed by the data object header and part of the first data packet. Figure 5.32 illustrates a detailed view of a typical data packet, which consists of a packet header and one or more payloads. The packet header contains a *timestamp* field, specifying the time (in milliseconds) when the packet should be sent out by the media server. Note that the timestamp of a later data packet is always larger than that of the previous packet.

Each payload in a data packet corresponds to a (part of) media object, e.g., a video frame, in a particular media stream. Each payload contains a header and real data. In the payload header, a *presentation time* (in milliseconds) is specified, which determines when the media object carried in the payload should be rendered by the media client to the end-user. Since a media object may span multiple payloads, and since different audio and video streams contained in a media file may be played back at different times, the presentation times are not always increasing.

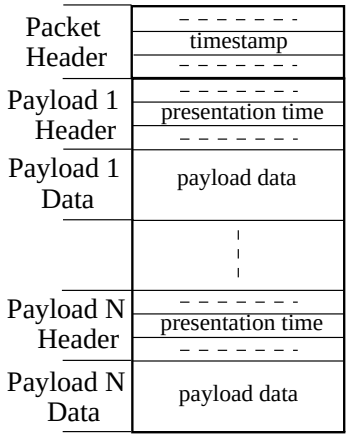



Figure 5.32: An ASF Data Packet

As shown in Figure 5.33, only four fields need to be modified for converging multiple linear media files. The first two are the *packet count* and the *playback duration* fields in the header object. These two fields of the first linear file are set to the largest values allowed to enable extension of the stream with data from subsequent files without confusing the client. The next two fields to be modified are the *timestamp* and the *presentation time* fields in every data packet. Note that each data packet contains only one timestamp but may contain multiple presentation times, each for a payload. These fields are modified by adding an offset equal to the corresponding time value in the last packet of the previous movie file.

As shown in Figure 5.34, when converging multiple linear media files into a non-linear file, for the first file, the *packet count* and the *playback duration* fields of the header object are modified. The index object of the first file is dropped by the SWORD client. For the following files, the SWORD client strips off the header objects and only modifies and forwards the data packets to the media client. The index object of all the following files, except that of the last file, will be dropped. The index object of the last file will be forwarded to the media client to announce the end of the entire non-linear media file.

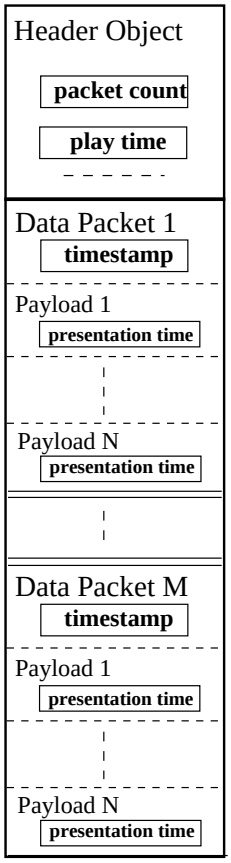



Figure 5.33: Fields Changed When Modifying an ASF File for Convergence

#### 5.4.2.3 SWORD Server Component Design

To enable non-linear media stream merging, the SWORD server must know the structure of each requested non-linear media file. Currently, these structures are hard-coded in the SWORD server. In the future, they could be input from a XML-like file during the initiation stage.

Similar to linear media stream merging, the SWORD server runs a non-linear media stream merging algorithm, specifically, a simple variant of HSM-UP, at regular intervals. In each round, for each non-linear media file, if the SWORD server finds out that a group of clients that is listening to a stream delivering portion $a$ of this file is not snooping on any previous stream delivering the same non-linear media file, it will search for a merge target stream for the client group. The merge target stream is the closest previous stream that is delivering the same portion $a$, or, if

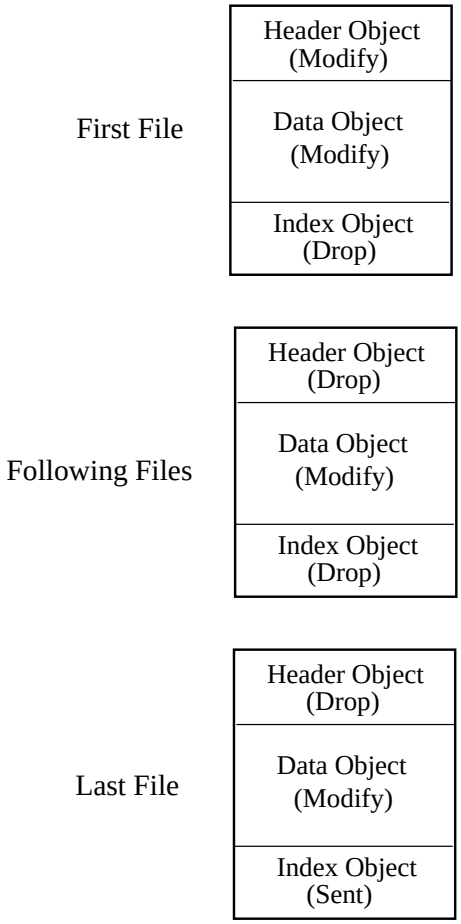



Figure 5.34: File Convergence at SWORD Client Component

all previous streams have past the branch point, the closest earlier stream that is delivering data from a portion in the sub-tree rooted at the portion $a$. If the SWORD server determines a merge target stream for the client group, it sends an UPDATE message, with the multicast address of the merge target stream, to each client in the group. Each client replies with a FIRST_GROUP message, indicating the first packet it has received from the merge target stream. The above logic, except the selection of merge target, is similar to stream merging of linear media. The difference lies in how the SWORD server reacts when an end-user selects the next portion at a branch point.

### Continuing

When the earliest client (in terms of playback progress) in a client group, which is listening to a stream delivering portion $a$, selects portion $b$ at a branch point, the SWORD server receives a request for that portion from the corresponding SWORD client. The SWORD server forwards the request to the media server. After receiving data from the media server, it delivers the data as a new multicast stream. At the same time, the SWORD server checks whether the stream that the group is currently snooping on is still a valid merge target. Then, the server sends an UPDATE message to each client in the group, letting it listen to the new multicast stream and continue or stop snooping on the current merge target stream. If the current merge target is no longer valid, the server will search for a new merge target for the group in the next round of the non-linear stream merging algorithm. Note that if some later client groups are snooping on the stream transmitting portion $a$, they will be informed by UPDATE messages to snoop on the new stream delivering portion $b$, since that stream is a valid merge target for these client groups.

### Branching

When the SWORD server receives a request for the next portion $b$ from a SWORD client $C$ that is not the earliest client in a group, the SWORD server checks whether the group to which client $C$ belongs had requested the same portion (the group currently may be receiving portion $b$ or a portion on a path the portion $b$ leads to). If it had, the SWORD server will suppress the request because the required data is already being acquired and buffered by client $C$. If the group had not requested portion $b$, the client $C$ is selecting a different path from that selected by all previous clients in the group, i.e., the client $C$ is *branching* out from the group. The SWORD server forwards the request for portion $b$ to the media server. After receiving data from the media server, it transmits this data in a new multicast stream, and then sends an UPDATE message to the SWORD client $C$, letting it listen to this new multicast stream and stop snooping on the current merge target stream. In the next

run of the stream merging algorithm, the SWORD server searches for a new merge target for this branched-out client $C$. The new multicast stream may also become the merge target of some client groups. Note that all the other clients in the original group will not be affected by the branch-out of client $C$.

### 5.4.3 Experiments with Non-linear Media Streaming

The purpose of the experiments in this section is to ensure that non-linear media streaming is working successfully in the SWORD system. For an end user that selects a playback path, the user should be able to view seamless playback at branch points, and should not realize that the playback is a concatenation of separate files. To verify that the system has achieved this goal, real users are invited to use the SWORD system, each with a separate machine. The users request the same non-linear media file from the system. Each user then watches the playback of a selected path (which have been hard-coded in the system) and judge whether a continuous and seamless playback is received and whether boundaries between subsequent linear media files are perceptible.

The SWORD server should carry out the implemented non-linear stream merging algorithm properly, including selecting the right merge target stream, and handling the *continuing* and the *branching* cases at a branch point in the way described in the previous section. To verify this, a trace-point is instrumented in the SWORD server. This trace-point records all UDP multicast data packets sent out from the SWORD server. By analyzing these packets, the start time, end time, and the rate of each multicast stream sent out from the server can be determined, and so the events related to stream merging occurances can be inferred. Of interest to this study are three types of events, namely the *merging*, *continuing*, and *branching* events. Three experiments, which will be described in detail later, are conducted to verify that the server has performed correctly in those three aspects.

As shown in Figure 5.35, the experiments were performed over a 100 Mbps Ethernet network, with one media server machine and three client machines. The media server machine is an IBM X252 server, with XEON 1.5 GHz MP processors with HyperThreading, 8 GB memory, and 12 36.4 GB 15K RPM Hot Swap disks connected to two UltraSCSI 160 controllers. Each client machine is an IBM IntelliStation M Pro desktop, with P4 2.8 GHz processor, 1 GB memory, and a 36.4 GB 15K RPM Ultra320 SCSI hard disk. The media server machine runs Windows 2000 Advanced Server, which by default enables Windows Media Service. Each client machine runs Windows XP Professional and Windows Media Player 9.0. The SWORD server component, and the three SWORD client components, each associated with an instance of Windows Media Player running on a client machine, are running on client machine 1.

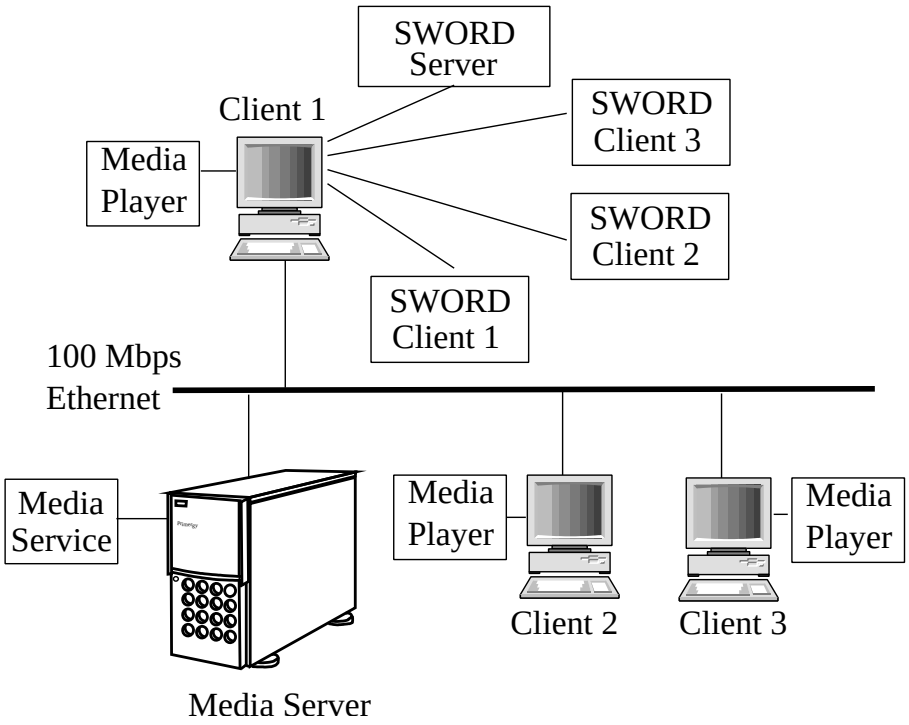



Figure 5.35: Experimental Environment and Set-up

In each experiment, three users, each using a client machine, initiate the Media Player, and then send a request for the same (non-linear) file. For all experiments, these users all reported that they obtained a continuous and seamless playback of

the videos on the specified playback path. The remainder of this section describes the three experiments and the analysis results of the data obtained from the trace point in the SWORD server.

### Experiment 1

The objective of this experiment is to demonstrate that the SWORD server can handle client *continuing* and client *branching out* at branch points correctly. As described in the previous section, if the earliest client in a group reaches a branch point, the server continues delivering the stream with the data on the next portion. If a client that is not the earliest client reaches a branch point and selects a different portion from the one(s) chosen by the earlier client(s), the server initiates a new stream with the requested data and sends it to the branched-out client.

Figure 5.36 shows the structure of the non-linear media file used in the first experiment. This non-linear media file is composed of seven media portions named $S0$ to $S6$. The number of data packets associated with each file is labeled on the corresponding tree link. The pre-specified path for client 1 is $S0$–$S1$–$S3$, for client 2 is $S0$–$S2$–$S5$, and for client 3 is $S0$–$S2$–$S6$. Note that although these paths are hard-coded in both the SWORD server and client, they are checked only when a SWORD client reaches a branch point, particularly in this study, when the SWORD client finishes transmitting data packets from the corresponding media portion[4], so as to simulate the scenario in which the SWORD server does not have any *a priori* knowledge of client path selection.

Figure 5.37 depicts how the SWORD server delivers UDP multicast streams in this experiment. The x-axis is the time, and the y-axis is the number of a transmitted packet, assuming that the packets of the media portions are concatenated and numbered in the way shown in the figure. Table 5.4 lists the events that occur at the marked times. At time $T3$, client 2 merges with client 1. The server thus terminates the stream sent to client 2. At time $T5$, client 3 merges with the group of clients 1

[4]In the current implementation, the client will send a TCP message to the SWORD server to announce that it has reached the branch point.

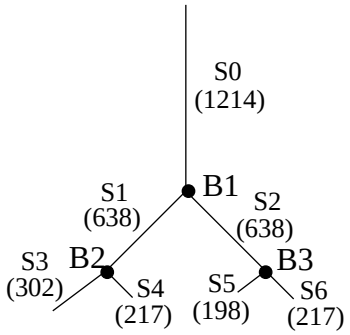



Figure 5.36: Non-linear Media File Used in Experiment 1

and 2. At time $T6$, client 1's playback reaches branch point $B1$. Since client 1 is the earliest client in the group, the SWORD server continues delivering new data on the same stream to the whole group. At time $T8$, client 2's playback reaches branch point $B1$. Since client 2 selects a different portion than the one chosen by client 1, it branches out from the group, and so the server sends out a new stream to client 2. At time $T9$, client 3 branches out from the group and receives a new stream from the server. The branch-out also happens at time $T16$ when client 3 selects a different portion than that chosen by client 2 at branch point $B3$.

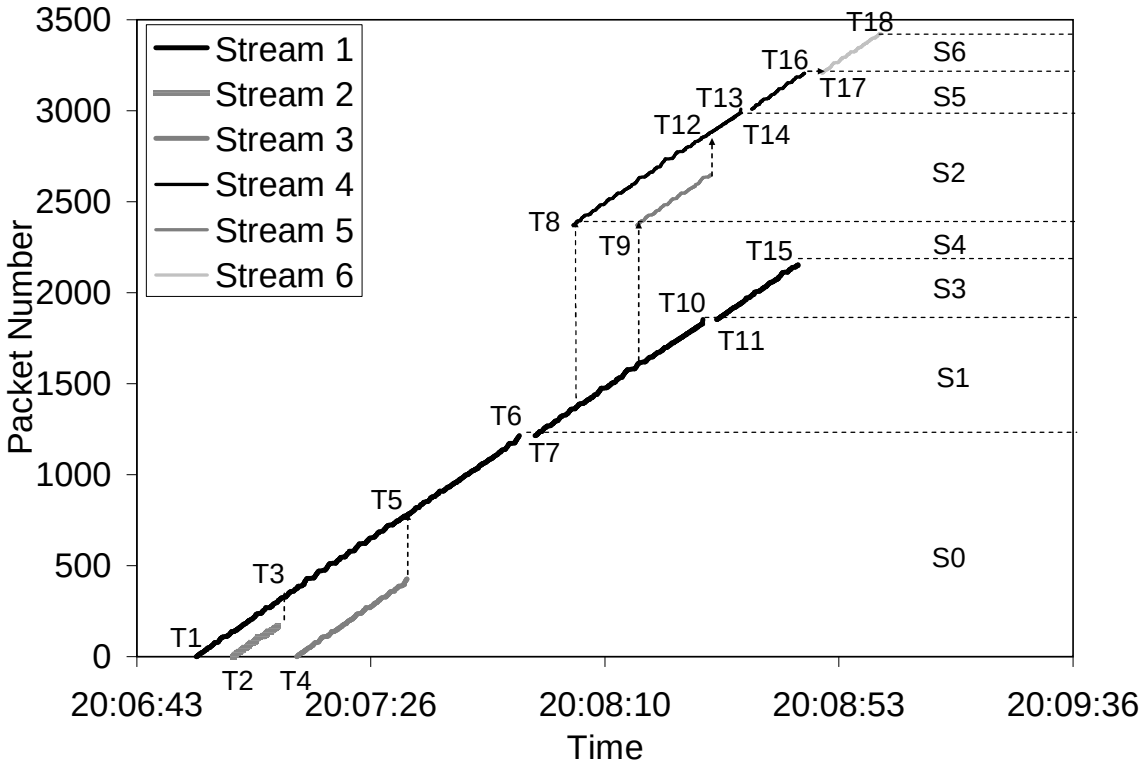



Figure 5.37: Experiment 1

Table 5.4: Events in Experiment 1

| Time | Event |
|---|---|
| $T1$ | The server starts stream 1 that delivers data on portion $S0$ upon receiving the request from client 1 |
| $T2$ | The server starts stream 2 that delivers data on portion $S0$ upon receiving the request from client 2 |
| $T3$ | The server terminates stream 2 because client 2 merges with client 1 |
| $T4$ | The server starts stream 3 that delivers data on portion $S0$ upon receiving the request from client 3 |
| $T5$ | The server terminates stream 3 because client 3 merges with the group of clients 1 and 2 |
| $T6$ | The server finishes delivering data of portion $S0$ on stream 1 |
| $T7$ | The server starts to deliver data of portion $S1$ on stream 1 upon client 1's selection at branch point $B1$ |
| $T8$ | The server starts stream 4 that delivers data on portion $S2$ upon client 2's selection at branch point $B1$ |
| $T9$ | The server starts stream 5 that delivers data on portion $S2$ upon client 3's selection at branch point $B1$ |
| $T10$ | The server finishes delivering data of portion $S1$ on stream 1 |
| $T11$ | The server starts to deliver data of portion $S3$ on stream 1 upon client 1's selection at branch point $B2$ |
| $T12$ | The server terminates stream 5 because client 3 merges with client 2 |
| $T13$ | The server finishes delivering data of portion $S2$ on stream 4 |
| $T14$ | The server starts to deliver data of portion $S5$ on stream 4 upon client 2's selection at branch point $B3$ |
| $T15$ | The server terminates stream 1 because it has finished transmitting portion $S3$ |
| $T16$ | The server terminates stream 4 because it has finished transmitting portion $S5$ |
| $T17$ | The server starts stream 6 that delivers data on portion $S6$ upon client 3's selection at branch point $B3$ |
| $T18$ | The server terminates stream 6 because it has finished transmitting portion $S6$ |

### Experiment 2

The objective of this experiment is to demonstrate that the server can utilize the non-linear media structure when determining a merge target. Figure 5.38 shows the structure of the non-linear media file used in the second experiment. The name of each portion and the associated number of data packets are labelled on the corresponding tree link. The pre-specified path for all clients is $S0$–$S1$–$S3$.

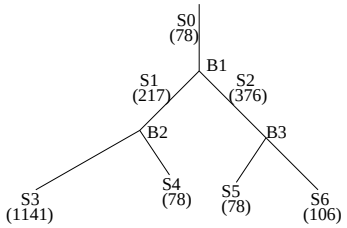



Figure 5.38: Non-linear Media File Used in Experiment 2

Figure 5.39 depicts how the SWORD server delivers UDP multicast streams in this case. Table 5.5 lists the events that occur at the marked times. At time $T4$, when client 2's request arrives at the SWORD server, the SWORD server has already been transmitting portion $S1$ on stream 1 to client 1. If the server used a linear media stream merging algorithm, it would not let client 2 snoop on stream 1 because it would think that portion $S1$ is a different file. Using the non-linear media stream merging algorithm, however, the server considers stream 1 as a valid merge target for client 2. For the same reason, at time $T10$, the server instructs client 3 to snoop on stream 1 even though the SWORD server has been delivering portion $S3$ when client 3 makes the request.

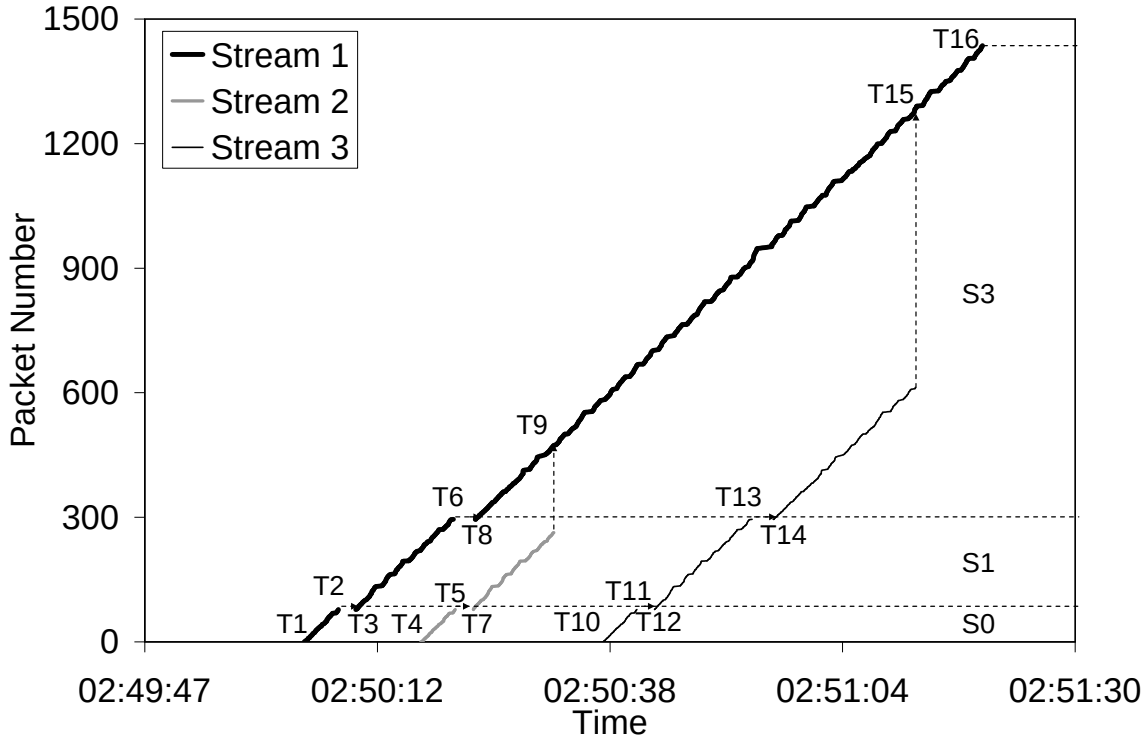



Figure 5.39: Experiment 2

Table 5.5: Events in Experiment 2

| Time | Event |
|---|---|
| $T1$ | The server starts stream 1 that delivers data on portion $S0$ upon receiving the request from client 1 |
| $T2$ | The server finishes delivering data of portion $S0$ on stream 1 |
| $T3$ | The server starts to deliver data of portion $S1$ on stream 1 upon client 1's selection at branch point $B1$ |
| $T4$ | The server starts stream 2 that delivers data on portion $S0$ upon receiving the request from client 2 |
| $T5$ | The server finishes delivering data of portion $S0$ on stream 2 |
| $T6$ | The server finishes delivering data of portion $S1$ on stream 1 |
| $T7$ | The server starts to deliver data of portion $S1$ on stream 2 upon client 2's selection at branch point $B1$ |
| $T8$ | The server starts to deliver data of portion $S3$ on stream 1 upon client 1's selection at branch point $B2$ |
| $T9$ | The server terminates stream 2 because client 2 merges with client 1 |
| $T10$ | The server starts stream 3 that delivers data on portion $S0$ upon receiving the request from client 3 |
| $T11$ | The server finishes delivering data of portion $S0$ on stream 3 |
| $T12$ | The server starts to deliver data of portion $S1$ on stream 3 upon client 3's selection at branch point $B1$ |
| $T13$ | The server finishes delivering data of portion $S1$ on stream 3 |
| $T14$ | The server starts to deliver data of portion $S3$ on stream 3 upon client 3's selection at branch point $B2$ |
| $T15$ | The server terminates stream 3 because client 3 merges with the group of clients 1 and 2 |
| $T16$ | The server terminates stream 1 because it has finished transmitting portion $S3$ |

### Experiment 3

The objective of this experiment is to demonstrate that because the server does not have advance knowledge of client path selection, the server may select a wrong merge target for a client. As a result, the client may miss a chance to listen to a stream from which it can receive useful data.

Figure 5.40 shows the structure of the non-linear media file used in the third experiment. The name of each portion and the associated number of data packets are labelled on the corresponding tree link. The pre-specified path for clients 1 and 3 is $S0$–$S1$–$S3$, and for client 2 is $S0$–$S2$–$S5$.

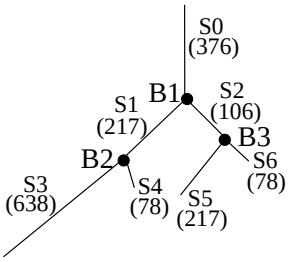



Figure 5.40: Non-linear Media File Used in Experiment 3

Figure 5.41 shows how the SWORD server delivers UDP multicast streams in the scenario of this experiment. Table 5.6 explains what has happened at the marked times. At time $T5$, Client 3 is instructed by the SWORD server to snoop on stream 2 because it is the closest stream that is transmitting data on the portion client 3 is listening to. The server does not know at that time that client 2 will take a different path than that taken by client 3 at branch point $B1$. At time $T12$, after branching out from the group at branch point $B1$, client 3 starts to snoop on stream 1, but it is too late for client 3 to catch up to stream 1 before that stream terminates. If the server knew that client 1 and client 3 were taking the same path, the server could let client 3 snoop on stream 1 from the time client 3 makes the request, and as a result client 3 might be able to merge with client 1. However, without having any *a priori* knowledge of client path selection, the server cannot avoid sometimes choosing wrong merge targets.

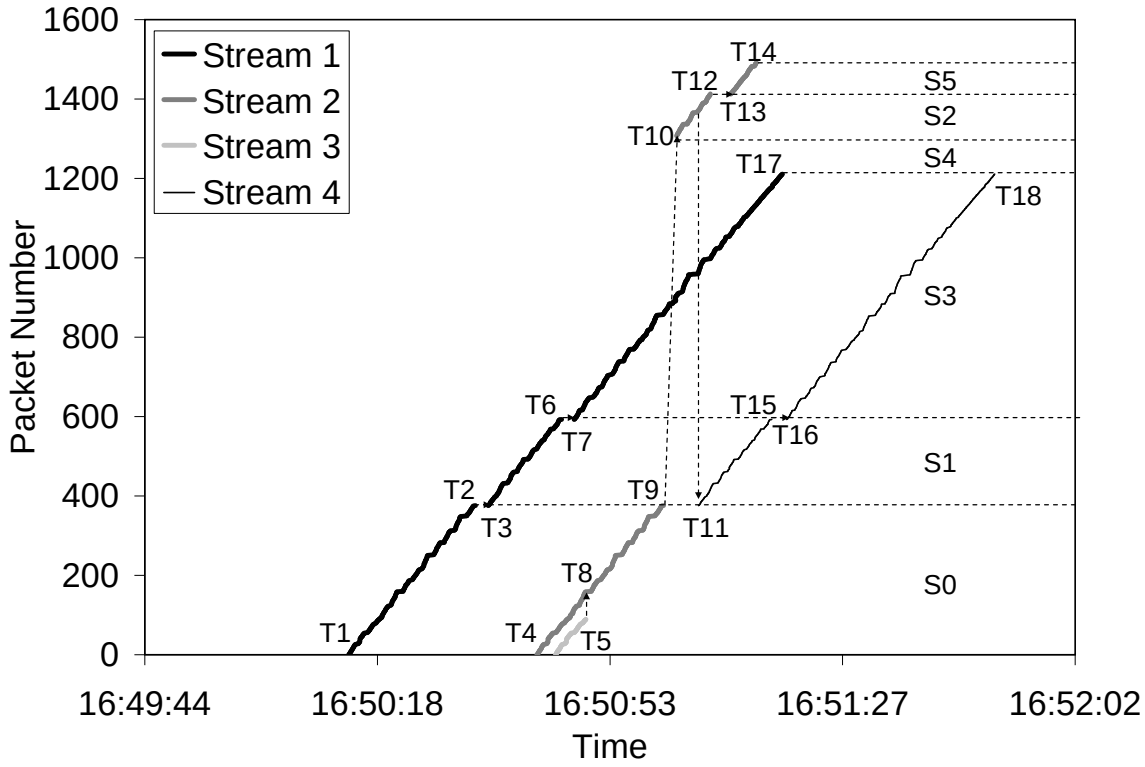



Figure 5.41: Experiment 3

Table 5.6: Events in Experiment 3

| Time | Event |
|---|---|
| $T1$ | The server starts stream 1 that delivers data on portion $S0$ upon receiving the request from client 1 |
| $T2$ | The server finishes delivering data of portion $S0$ on stream 1 |
| $T3$ | The server starts to deliver data of portion $S1$ on stream 1 upon client 1's selection at branch point $B1$ |
| $T4$ | The server starts stream 2 that delivers data on portion $S0$ upon receiving the request from client 2 |
| $T5$ | The server starts stream 3 that delivers data on portion $S0$ upon receiving the request from client 3 |
| $T6$ | The server finishes delivering data of portion $S1$ on stream 1 |
| $T7$ | The server starts to deliver data of portion $S3$ on stream 1 upon client 1's selection at branch point $B2$ |
| $T8$ | The server terminates stream 3 because client 3 merges with client 2 |
| $T9$ | The server finishes delivering data of portion $S0$ on stream 2 |
| $T10$ | The server starts to deliver data of portion $S2$ on stream 2 upon client 2's selection at branch point $B1$ |
| $T11$ | The server starts stream 4 that delivers data on portion $S1$ upon client 3's selection at branch point $B1$ |
| $T12$ | The server finishes delivering data of portion $S2$ on stream 2 |
| $T13$ | The server starts to deliver data of portion $S5$ on stream 2 upon client 2's selection at branch point $B3$ |
| $T14$ | The server terminates stream 2 because it has finished transmitting portion $S5$ |
| $T15$ | The server finishes delivering data of portion $S1$ on stream 4 |
| $T16$ | The server starts to deliver data of portion $S3$ on stream 4 upon client 3's selection at branch point $B2$ |
| $T17$ | The server terminates stream 1 because it has finished transmitting portion $S3$ |
| $T18$ | The server terminates stream 4 because it has finished transmitting portion $S3$ |

## 5.5 Summary

In this chapter, an advanced type of media termed *non-linear* media is proposed. Contrary to traditional *linear* media files that contain only a single sequence of data units, non-linear media files contain parallel sequences of data units, among which clients can dynamically select at designated branch points. Non-linear media may enable more customized, personalized, and interactive content delivery services. This chapter investigates the design and performance issues in scalable on-demand streaming of such media. Lower bounds on the minimum required server bandwidth for various non-linear media scalable on-demand streaming approaches are derived, practical non-linear media scalable delivery protocols are developed, and, as a proof-of-concept, a simple scalable delivery protocol is implemented in a non-linear media streaming prototype system.

# Chapter 6

# Conclusions

Scalable streaming can allow newly emerged multimedia streaming applications, such as video-on-demand and distance learning, to be widely deployed and cost-effectively operated.

In current streaming systems, a separate stream is dedicated to each client request, which is not sufficiently scalable for wide-area delivery of popular and bandwidth-intensive multimedia files to potentially large numbers of concurrent clients. To address this problem, previous research has developed efficient delivery protocols. These protocols share a server stream among clients that make closely-spaced requests for the same file by either broadcasting popular files (*periodic broadcast protocols*) or dynamically aggregating individual clients into groups and multicasting to each group (*immediate service protocols*). Previous research has found that these protocols can achieve server bandwidth that grows much slower than linearly with the file request rate, and with the inverse of client start-up delay.

This thesis addresses the protocol design and performance issues in scalable on-demand streaming of stored complex multimedia. It contains three parts: *network bandwidth analysis for scalable on-demand streaming*, *scalable on-demand streaming of VBR media*, and *scalable on-demand streaming of non-linear media*. Section 6.1 summarizes each part of the thesis work, Section 6.2 states the main contributions, and Section 6.3 briefly outlines further research directions.

## 6.1 Thesis Summary

### Network Bandwidth Analysis for Scalable On-demand Streaming

Previous research has analyzed server bandwidth requirements of scalable on-demand streaming. However, no previous research has quantified network bandwidth savings. The achieved reduction in network bandwidth may differ from that for server bandwidth if delivery is over point-to-point networks (e.g., the Internet), which must rely on multicast to realize one-to-many transmission. The first part of this thesis investigates the network bandwidth requirement for scalable on-demand streaming in this case. Tight bounds on the minimum required network bandwidth have been developed, first for simple canonical multicast delivery tree topologies, and then for delivery trees constructed by alternative algorithms over randomly generated network topologies and client locations. The results suggest that in practical cases, the minimum required network bandwidth for scalable on-demand streaming typically scales as $K/\ln(K)$ as the number of client sites $K$ increases for a fixed request rate per client site, and as $\ln\left(\frac{N}{ND+1}\right)$ as the total file request rate $N$ increases or client start-up delay $D$ decreases, for fixed number of client sites. Multicast delivery trees configured to minimize network bandwidth usage rather than latency are found to reduce the minimum required network bandwidth only modestly. Furthermore, it is possible to achieve close to the minimum possible network and server bandwidth usage simultaneously with practical scalable delivery protocols.

### Scalable On-demand Streaming of VBR media

VBR-encoded files have the advantage of enabling efficient support of constant playback quality. Work-ahead smoothing is a key technology for reducing the rate variability of VBR files. However, in the context of scalable on-demand streaming, it fundamentally conflicts with scalable delivery protocols with respect to bandwidth usage. Previous work has investigated the impact of work-ahead smoothing on the bandwidth efficiency of periodic broadcast protocols. The second part of this thesis studies the interaction between work-ahead smoothing and immediate service pro-

tocols. A lower bound on the server bandwidth requirement for scalable on-demand streaming of VBR media is derived. The lower bound analysis reveals that the straightforward approach that directly applies immediate service protocols designed for CBR media to a work-ahead smoothed VBR file consumes more server bandwidth than delivering the VBR file as is. This result motivates the design of a new immediate service protocol termed VBR bandwidth skimming (VBRBS) that uses constant bit rate streaming, when sufficient client storage is available, yet fruitfully exploits the knowledge of a VBR profile. Simulation results show that VBRBS can considerably reduce server bandwidth usage from that of the straightforward approach.

**Scalable On-demand Streaming of Non-linear media**

The multimedia files delivered in current streaming systems are *linear* in that they contain a single sequence of data units, and all clients requesting the same file always obtain all or a sub-sequence of the same content. Network delivery may enable *non-linear* media containing parallel sequences of data units, among which clients can dynamically select at designated branch points. Examples of non-linear media are pick-your-own-ending movies, analogous to pick-your-own-ending children's books. The third part of this thesis investigates the design and performance issues in scalable on-demand streaming of such media. Lower bounds on the minimum required server bandwidth for various non-linear media scalable on-demand streaming approaches are derived, practical non-linear media scalable delivery protocols are developed, and, as a proof-of-concept, a simple scalable delivery protocol is implemented in a non-linear media streaming prototype system.

## 6.2 Thesis Contributions

In summary, the main contributions of this thesis are:

- Tight bounds on the minimum required network bandwidth for scalable on-demand streaming of constant bit rate (CBR) linear media files are developed. These bounds can be easily extended to variable bit rate (VBR) and non-linear media files.

- A lower bound on the minimum required server bandwidth for scalable on-demand streaming of VBR media is developed. A scalable immediate service delivery protocol for such media, VBR bandwidth skimming (VBRBS), is proposed and its performance evaluated.

- Lower bounds on the minimum required server bandwidth for various non-linear media scalable on-demand streaming approaches are developed. Practical non-linear media periodic broadcast and immediate service delivery protocols are designed and their performance evaluated. As a proof-of-concept, a simple non-linear media immediate service delivery protocol has been implemented in the SWORD prototype system.

## 6.3 Future Work

There are many open research issues related to the work presented in this thesis. Some possible future research directions include:

- Designing *hybrid* scalable delivery approaches for non-linear media on-demand streaming. The analysis results in Section 5.2 have shown that popularity-based delivery approaches work well for branch points having very skewed branch selection probabilities, but not as efficiently with respect to server bandwidth usage as the *next* approach at branch points with similar branch selection probabilities. However, the *next* approach incurs more client data

overhead than popularity-based approaches. Hybrid approaches may be developed that consume less server bandwidth than popularity-based approaches and less client data overhead than the *next* approach. For example, considering a branch point at which some branches are popular (with selection probabilities greater than a threshold) and the others are much less popular (with selection probabilities less than the threshold), clients could snoop on multicasts from all non-popular branches (like *next*), and also predict a path that includes a popular branch (like *pred*) and snoop on multicasts of portions on the path. Preliminary experiments have found that this hybrid of *next* and *pred* may perform even better than either approach in some cases.

- Improving the design of the non-linear media periodic broadcast protocols derived in Section 5.3.2. The OPB-UP protocol assumes simply that a client starts the reception of a new segment immediately after the reception of a previous segment has completed. In some cases, delaying the reception of a segment until after the choice of the next branch has been made may be more beneficial with respect to bandwidth usage. The OPB protocols have made a number of other assumptions on non-linear media streaming. New variants of the OPB protocol may be devised for more general non-linear media structures. Moreover, the OPB protocols should be more robust to recover from mispredicted paths.

- Adding interactive function support in non-linear media stream merging protocols. Interactive functions such as pause, fast-forward, and rewind are necessary to make on-demand streaming applications (e.g., video-on-demand) appealing to end users and competitive to video rental. Interactive functions may be realized by re-allocating a new stream to each client that initiated the request. In the context of non-linear media streaming, how these new streams interact with non-linear media stream merging algorithms needs to be investigated.

- Continuing the development of the non-linear media on-demand streaming system based on the SWORD prototype. Possible future work includes creating real non-linear media files using existing multimedia authoring tools such as Adobe Premier, implementing the user interface through which end-users can make selections at branch points, defining a data format for non-linear media files, implementing non-linear media periodic broadcast protocols in the prototype, exploring alternative interaction protocols between the SWORD server and client to realize non-linear media stream merging, and improving the reliability and robustness of non-linear media streaming.

- Conducting experiments on local-area and wide-area delivery of non-linear media files in the SWORD prototype. In the thesis study, preliminary experiments on non-linear media stream merging have been conducted in a LAN environment. It would be useful to conduct more experiments over local-area networks and over the Internet, possibly using PlanetLab [92], a globally distributed platform for conducting Internet-scale network experiments. The focuses of the experimentation could be on understanding the overhead of multicast join/leave, the overhead of non-linear media stream merging algorithms, the robustness of non-linear media streaming protocols, and the efficiency of non-linear media streaming over an application-level multicast setting.

# References


[1] C. C. Aggarwal, J. L. Wolf, and P. S. Yu. On optimal batching policies for video-on-demand storage servers. In *Proceedings of the 1996 IEEE International Conference on Multimedia Computing and Systems (ICMCS'96)*, pages 253–282, Hiroshima, Japan, June 1996.

[2] C. C. Aggarwal, J. L. Wolf, and P. S. Yu. On optimal piggyback merging policies for video-on-demand systems. In *Proceedings of the 1996 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'96)*, pages 200–209, Philadelphia, PA, May 1996.

[3] C. C. Aggarwal, J. L. Wolf, and P. S. Yu. A permutation-based pyramid broadcasting scheme for video-on-demand systems. In *Proceedings of the 1996 IEEE International Conference on Multimedia Computing and Systems (ICMCS'96)*, pages 118–126, Hiroshima, Japan, June 1996.

[4] Akamai, July 2004. `http://www.akamai.com`.

[5] J. M. Almeida, D. L. Eager, M. Ferris, and M. K. Vernon. Provisioning content distribution networks for streaming media. In *Proceedings of the* $21^{st}$ *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'02)*, pages 1746–1755, New York, NY, June 2002.

[6] J. M. Almeida, D. L. Eager, and M. K. Vernon. A hybrid caching strategy for streaming media files. In *Proceedings of the 2001 IS&T/SPIE Conference on Multimedia Computing and Networking (MMCN'01)*, pages 200–212, San Jose, CA, January 2001.

[7] J. M. Almeida, J. Krueger, D. L. Eager, and M. K. Vernon. Analysis of educational media server workloads. In *Proceedings of the* $11^{th}$ *International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'01)*, pages 21–30, Port Jefferson, NY, June 2001.

[8] K. C. Almeroth. The evolution of multicast: From the MBone to interdomain multicast to Internet2 deployment. *IEEE Network*, 14(1):10–20, January/February 2000.

[9] K. C. Almeroth and M. H. Ammar. The use of multicast delivery to provide a scalable and interactive video-on-demand service. *IEEE Journal on Selected Areas in Communications*, 14(6):1110–1122, August 1996.

[10] M. H. Ammar and J. W. Wong. On the optimality of cyclic transmission in teletex systems. *IEEE Transactions on Communications*, 7(6):68–73, January 1987.

[11] Apple QuickTime, July 2004. http://www.apple.com/quicktime/.

[12] T. Ballardie, P. Francis, and J. Crowcroft. Core based trees (CBT). In *Proceedings of the 1993 ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'93)*, pages 85–95, San Francisco, CA, September 1993.

[13] S. Banerjee, C. Kommareddy, K. Kar, S. Bhattacharjee, and S. Khuller. Construction of an efficient overlay multicast infrastructure for real-time applications. In *Proceedings of the* $22^{nd}$ *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'03)*, pages 1521–1531, San Francisco, CA, March/April 2003.

[14] A. Bar-Noy, J. Goshi, R. E. Ladner, and K. Tam. Comparison of stream merging algorithms for media-on-demand. *Multimedia Systems*, 9(5):411–423, March 2004.

[15] A. Bar-Noy and R. E. Ladner. Efficient algorithms for optimal stream merging for media-on-demand. http://www.cs.washington.edu/homes/ladner/papers.html, May 2001.

[16] T. Bates, R. Chandra, D. Katz, and Y. Rekhter. Multiprotocol extensions for BGP-4. RFC 2283, February 1998.

[17] P. Bernier. Cablecos: Your video wish is our command. Xchange Magazine, January 2003. http://www.xchangemag.com/articles/311coverstory.html.

[18] S. Bhattacharyya. An overview of source-specific multicast (SSM). RFC 3569, July 2003.

[19] Y. Birk and R. Mondri. Tailored transmissions for efficient near-video-on-demand service. In *Proceedings of the 1999 IEEE International Conference on Multimedia Computing and Systems (ICMCS'99)*, pages 226–231, Florence, Italy, June 1999.

[20] Y. Cai, K. A. Hua, and K. Yu. Optimizing patching performance. In *Proceedings of the 1999 IS&T/SPIE Conference on Multimedia Computing and Networking (MMCN'99)*, pages 204–215, San Jose, CA, January 1999.

[21] K. Calvert and E. W. Zegura. GT internetwork topology models (GT-ITM), July 2004. http://www.cc.gatech.edu/fac/Ellen.Zegura/gt-itm/gt-itm.tar.gz.

[22] S. W. Carter and D. D. E. Long. Improving video-on-demand server efficiency through stream tapping. In *Proceedings of the* $6^{th}$ *International Conference on Computer Communications and Networks (ICCCN'97)*, pages 200–207, Las Vegas, NV, September 1997.

[23] Y. Chawathe, S. McCanne, and E. A. Brewer. An architecture for Internet content distribution as an infrastructure service. `http://yatin.chawathe.com/~yatin/papers/scattercast.ps`, February 2000.

[24] Y. Chu, S. Rao, and H. Zhang. The case for end system multicast. In *Proceedings of the 2000 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'00)*, pages 1–12, Santa Clara, CA, June 2000.

[25] J. C.-I. Chuang and M. A. Sirbu. Pricing multicast communication: A cost-based approach. *Telecommunication Systems*, 17(3):281–297, July 2001.

[26] Cisco voice and IP communications, July 2004. `http://www.cisco.com/en/US/products/sw/voicesw/index.html`.

[27] E. G. Coffman, P. Jelenkovic, and P. Momcilovic. The dyadic stream merging algorithm. *Journal of Algorithms*, 43(1):120–137, April 2002.

[28] A. Dan and D. Sitaram. A generalized interval caching policy for mixed interactive and long video environments. In *Proceedings of the 1996 IS&T/SPIE Conference on Multimedia Computing and Networking (MMCN'96)*, pages 344–351, San Jose, CA, January 1996.

[29] A. Dan, D. Sitaram, and P. Shahabuddin. Scheduling policies for an on-demand video server with batching. In *Proceedings of the* $2^{nd}$ *ACM International Multimedia Conference (MULTIMEDIA'94)*, pages 15–23, San Francisco, CA, October 1994.

[30] A. Dan, D. Sitaram, and P. Shahabuddin. Dynamic batching policies for an on-demand video server. *Multimedia Systems*, 4(3):112–121, June 1996.

[31] S. Deering. Host extensions for IP multicasting. RFC 1112, August 1989.

[32] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, and L. Wei. The PIM architecture for wide area multicasting. *IEEE/ACM Transactions on Networking*, 4(2):153–162, April 1996.

[33] Digital Island, July 2004. `http://www.sandpiper.net/`.

[34] H. D. Dykeman, M. H. Ammar, and J. W. Wong. Scheduling algorithms for videotex systems under broadcast delivery. In *Proceedings of the 1986 IEEE International Conference on Communications (ICC'86)*, pages 1847–1851, Toronto, ON, Canada, June 1986.

[35] D. L. Eager and M. K. Vernon. Dynamic skyscraper broadcasts for video-on-demand. In *Proceedings of the* $4^{th}$ *International Workshop on Advances in Multimedia Information Systems (MIS'98)*, pages 18–32, Istanbul, Turkey, September 1998.

[36] D. L. Eager, M. K. Vernon, and J. Zahorjan. Optimal and efficient merging schedules for video-on-demand servers. In *Proceedings of the* $7^{th}$ *ACM International Multimedia Conference (MULTIMEDIA'99)*, pages 199–202, Orlando, FL, November 1999.

[37] D. L. Eager, M. K. Vernon, and J. Zahorjan. Bandwidth skimming: A technique for cost-effective video-on-demand. In *Proceedings of the 2000 IS&T/SPIE Conference on Multimedia Computing and Networking (MMCN'00)*, pages 206–215, San Jose, CA, January 2000.

[38] 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):742–757, September/October 2001.

[39] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei. Protocol independent multicast – sparse mode (PIM-SM): Protocol specification. RFC 2362, June 1998.

[40] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On Power-Law relationships of the Ineternet topology. In *Proceedings of the 1999 ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'99)*, pages 251–262, Cambridge, MA, August 1999.

[41] W. Feng. *Video-on-demand Services: Efficient Transportation and Decompression of Variable-bit-rate Video*. Ph.D. dissertation, University of Michigan, 1996.

[42] W. Feng and J. Rexford. A comparison of bandwidth smoothing techniques for the transmission of prerecorded compressed video. In *Proceedings of the* $16^{th}$ *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'97)*, pages 58–66, Kobe, Japan, April 1997.

[43] B. Fenner, M. Handley, H. Holbrook, and I. Kouvelas. Protocol independent multicast – sparse mode (PIM-SM): Protocol specification (revised). IETF PIM WG Internet-Draft, `http://www.ietf.org/internet-drafts/draft-ietf-pim-sm-v2-new-09.txt`, February 2004.

[44] B. Fenner and D. Meyer. Multicast source discovery protocol (MSDP). RFC 3618, October 2003.

[45] L. Gao, J. Kurose, and D. Towsley. Efficient schemes for broadcasting popular videos. In *Proceedings of the* $8^{th}$ *International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'98)*, pages 317–329, Cambridge, UK, July 1998.

[46] L. Gao and D. Towsley. Supplying instantaneous video-on-demand services using controlled multicast. In *Proceedings of the 1999 IEEE International Conference on Multimedia Computing and Systems (ICMCS'99)*, pages 117–121, Florence, Italy, June 1999.

[47] L. Gao, Z. Zhang, and D. Towsley. Catching and selective catching: Efficient latency reduction techniques for delivering continuous multimedia. In *Proceedings of the $7^{th}$ ACM International Multimedia Conference (MULTIMEDIA'99)*, pages 203–206, Orlando, FL, October/November 1999.

[48] M. R. Garey, R. L. Graham, and D. S. Johnson. The complexity of computing Steiner minimal tree. *SIAM Journal on Applied Mathematics*, 34:477–495, 1978.

[49] M. N. Garofalakis, Y. E. Ioannidis, and B. Ozden. Resource scheduling for composite multimedia objects. In *Proceedings of the $24^{th}$ Very Large Data Bases Conference (VLDB'98)*, pages 74–85, New York, NY, August 1998.

[50] M. Garrett and W. Willinger. Analysis, modeling and generation of self-similar VBR video traffic. In *Proceedings of the 1994 ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'94)*, pages 269–280, London, England, August 1994.

[51] L. Goleniewski. *Telecommunications Essentials*. Addison-Wesley, 2002.

[52] L. Golubchik, J. Lui, and R. Muntz. Reducing I/O demand in video-on-demand storage servers. In *Proceedings of the 1995 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'95)*, pages 25–36, Ottawa, Canada, May 1995.

[53] B. Haberman and D. Thaler. Unicast-prefix-based IPv6 multicast addresses. RFC 3306, August 2002.

[54] F. Halsall. *Multimedia Communications: Applications, Networks, Protocols and Standards*. Addison-Wesley, 2001.

[55] H. Holbrook and D.R. Cheriton. IP multicast channels : EXPRESS support for large-scale single-source applications. In *Proceedings of the 1999 ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'99)*, pages 65–78, Cambridge, MA, August/September 1999.

[56] A. Hu. Video-on-demand broadcasting protocols: A comprehensive study. In *Proceedings of the $20^{th}$ Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'01)*, pages 508–517, Anchorage, AL, April 2001.

[57] A. Hu, I. Nikolaidis, and P. Beek. On the design of efficient video-on-demand broadcast schemes. In *Proceedings of* $7^{th}$ *International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'99)*, pages 262–270, College Park, MD, March 1999.

[58] K. A. Hua, Y. Cai, and S. Sheu. Patching: A multicast technique for true video-on-demand services. In *Proceedings of the* $6^{th}$ *ACM International Multimedia Conference (MULTIMEDIA'98)*, pages 191–200, Bristol, U.K., September 1998.

[59] K. A. Hua and S. Sheu. Skyscraper broadcasting: A new broadcasting scheme for metropolitan video-on-demand systems. In *Proceedings of the 1997 ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'97)*, pages 89–100, Cannes, France, September 1997.

[60] Internet Assigned Numbers Authority (IANA). Internet multicast addresses, May 2004. `http://www.iana.org/assignments/multicast-addresses`.

[61] R. Janakiraman, M. Waldvogel, and L. Xu. Fuzzycast: Efficient video-on-demand over multicast. In *Proceedings of the* $21^{st}$ *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'02)*, pages 920–929, New York, NY, June 2002.

[62] J. Jannotti, D. K. Gifford, and K. L. Johnson. Overcast: Reliable multicast with an overlay network. In *Proceedings of the* $3^{rd}$ *Symposium on Operating Systems Design and Implementation (OSDI'00)*, pages 197–212, San Diego, CA, October 2000.

[63] C. Jin, Q. Chen, and S. Jamin. Inet: Internet topology generator. Technical Report UM-CSE-TR-433-00, University of Michigan, 2000.

[64] JPEG Homepage, July 2004. `http://www.jpeg.org/jpeg/index.html`.

[65] L. Juhn and L. Tseng. Harmonic broadcasting for video-on-demand service. *IEEE Transactions on Broadcasting*, 43(3):268–271, September 1997.

[66] J. Kangasharju, J. Roberts, and K. W. Ross. Object replication strategies in content distribution networks. *Computer Communications*, 25(4):376–383, March 2002.

[67] M. Karlsson and M. Mahalingam. Do we need replica placement algorithms in content delivery networks? In *Proceedings of the* $7^{th}$ *International Web Content Caching and Distribution Workshop (WCW'02)*, Boulder, CO, August 2002.

[68] Kazaar, July 2004. `http://www.kazaar.de.tc/`.

[69] W. Ke, P. Basu, and T. D. C. Little. Time-domain modeling of batching under user interaction and dynamic adaptive piggybacking. In *Proceedings of the 2002 IS&T/SPIE Conference on Multimedia Computing and Networking (MMCN'02)*, pages 130–141, San Jose, CA, January 2002.

[70] L. Kou, G. Markowsky, and L. Berman. A fast algorithm for Steiner trees. *Acta Informatica*, 15:141–145, 1981.

[71] J. Kurose and K. Ross. *Computer Networking: A Top-down Approach Featuring the Internet*. Addison-Wesley, $2^{nd}$ edition, 2002.

[72] T. V. Lakshman, A. Ortega, and A. R. Reibman. VBR video: Trade-offs and potentials. *Proceedings of the IEEE*, 86(5):952–973, May 1998.

[73] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson. On the self-similar nature of Ethernet traffic (extended version). *IEEE/ACM Transactions on Networking*, 2(1):1–15, February 1994.

[74] F. Li and I. Nikolaidis. Trace-adaptive fragmentation for periodic broadcast of VBR video. In *Proceedings of the $9^{th}$ International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'99)*, pages 253–264, Basking Ridge, NJ, June 1999.

[75] F. Li and I. Nikolaidis. SCB: Staircase broadcast for media-on-demand systems. In *Proceedings of the $7^{th}$ International Workshop on Mobile Multimedia Communications (MoMuC'00)*, pages (3B–1–1)–(3B–1–6), Tokyo, Japan, October 2000.

[76] A. Mahanti, D. L. Eager, M. K. Vernon, and D. Sundaram-Stukel. Scalable on-demand media streaming with packet loss recovery. *IEEE/ACM Transactions on Networking*, 11(2):195–209, April 2003.

[77] D. Makaroff. *The Design, Implementation and Evaluation of a Variable Bit Rate Continuous Media File Server for a High-speed Network*. Ph.D. dissertation, University of British Columbia, July 1998.

[78] J. McManus and K. Ross. Video-on-demand over ATM: Constant-rate transmission and transport. *IEEE Journal on Selected Areas in Communication*, 14(6):1087–1098, August 1996.

[79] Microsoft Windows Media 9 series audio and video codecs, July 2004. `http://www.microsoft.com/windows/windowsmedia/9series/codecs.aspx`.

[80] Microsoft Windows Media 9 series, July 2004. `http://www.microsoft.com/windows/windowsmedia/default.aspx`.

[81] J. Moy. Multicast extensions to OSPF. RFC 1584, March 1994.

[82] MP3.com, July 2004. `http://www.mp3.com`.

[83] MPEG-4 overview, March 2002. `http://www.chiariglione.org/mpeg/standards/mpeg-4/mpeg-4.htm`.

[84] I. Nikolaidis, F. Li, and A. Hu. An inherently loss-less and bandwidth-efficient periodic broadcast scheme for VBR video. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'00)*, pages 116–117, Santa Clara, CA, June 2000.

[85] J.-P. Nussbaumer, B. V. Patel, F. Schaffa, and J. R. G. Sterbenz. Networking requirements for interactive video on demand. *IEEE Journal on Selected Areas in Communications*, 13(5):779–787, June 1995.

[86] J.-F. Pâris. A broadcasting protocol for compressed video. In *Proceedings of EUROMEDIA'99*, pages 78–84, Munich, Germany, April 1999.

[87] J.-F. Pâris, S. W. Carter, and D. D. E. Long. Efficient broadcasting protocols for video on demand. In *Proceedings of the* $6^{th}$ *International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'98)*, pages 127–132, Montreal, Canada, July 1998.

[88] J.-F. Pâris, S. W. Carter, and D. D. E. Long. A low bandwidth broadcasting protocol for video on demand. In *Proceedings of the* $7^{th}$ *International Conference on Computer Communications and Networks (ICCCN'98)*, pages 690–697, Lafayette, LA, October 1998.

[89] J.-F. Pâris, S. W. Carter, and D. D. E. Long. A hybrid broadcasting protocol for video on demand. In *Proceedings of the 1999 IS&T/SPIE Conference on Multimedia Computing and Networking (MMCN'99)*, pages 317–326, San Jose, CA, January 1999.

[90] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. ALMI: An application level multicast infrastructure. In *Proceedings of the* $3^{rd}$ *USENIX Symposium on Internet Technologies (USITS'01)*, pages 49–60, San Francisco, CA, March 2001.

[91] G. Phillips and S. Shenker. Scaling of multicast trees: Comments on the Chuang-Sirbu scaling law. In *Proceedings of the 1999 ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'99)*, pages 41–51, Cambridge, MA, August/September 1999.

[92] Planetlab, July 2004. `http://www.planet-lab.org/`.

[93] L. Qiu, V. Padmanabhan, and G. Voelker. On the placement of web server replicas. In *Proceedings of the* $20^{th}$ *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'01)*, pages 1587–1596, Anchorage, AL, April 2001.

[94] P. Radoslavov, R. Govindan, and D. Estrin. Topology-informed Internet replica placement. In *Proceedings of the* $6^{th}$ *International Workshop on Web Caching and Content Distribution (WCW'01)*, pages 384–392, Boston, MA, June 2001.

[95] S. Ranjan, R. Karrer, and E. W. Knightly. Wide area redirection of dynamic content by Internet data centers. In *Proceedings of the* $23^{rd}$ *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'04)*, Hong Kong, China, March 2004.

[96] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Topologically-aware overlay construction and server selection. In *Proceedings of the* $21^{st}$ *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'02)*, pages 1190–1199, New York, June 2002.

[97] RealNetworks Real Player, July 2004. `http://www.real.com/`.

[98] O. Rose. Statistical properties of MPEG video traffic and their impact on traffic modeling in ATM systems. In *Proceedings of the* $20^{th}$ *IEEE Conference on Local Computer Networks (LCN'95)*, pages 397–406, Minneapolis, MN, October 1995.

[99] J. D. Salehi, Z. Zhang, J. Kurose, and D. Towsley. Supporting stored video: Reducing rate variability and end-to-end resource requirement through optimal smoothing. In *Proceedings of the 1996 ACM SIGMETRICS International Conference on Measurement and Modeling of Computing Systems (SIGMETRICS'96)*, pages 222–231, Philadelphia, PA, May 1996.

[100] D. Saparilla, K. Ross, and M. Reisslein. Periodic broadcasting with VBR-encoded video. In *Proceedings of the* $18^{th}$ *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'99)*, pages 464–471, New York, NY, March 1999.

[101] S. Sen, L. Gao, J. Rexford, and D. Towsley. Optimal patching schemes for efficient multimedia streaming. In *Proceedings of the* $9^{th}$ *International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'99)*, pages 265–277, Basking Ridge, NJ, June 1999.

[102] S. Sen, L. Gao, and D. Towsley. Frame-based periodic broadcast and fundamental resource tradeoffs. Technical Report 99-78, University of Massachusetts-Amherst, 1999.

[103] S. Sen, J. Rexford, and D. Towsley. Proxy prefix caching for multimedia streams. In *Proceedings of the* $18^{th}$ *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'99)*, pages 1310–1319, New York, NY, March 1999.

[104] Shaw Video-on-demand, July 2004. https://secure.shaw.ca/sod/home.asp.

[105] Short MPEG-1 description, June 1996. http://www.chiariglione.org/mpeg/standards/mpeg-1/mpeg-1.htm.

[106] Short MPEG-2 description, October 2000. http://www.chiariglione.org/mpeg/standards/mpeg-2/mpeg-2.htm.

[107] J. Song, A. Dan, and D. Sitaram. Efficient retrieval of composite multimedia objects in the JINSIL distributed system. In *Proceedings of the 1997 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'97)*, pages 260–271, Seattle, WA, June 1997.

[108] Streaming radio and television stations, July 2004. http://www.org.mk/radio/.

[109] D. Sundaram-Stukel, M. K. Vernon, J. Zahorjan, D. L. Eager, and B. Schwartz. SWORD prototype. Internal report.

[110] A. S. Tanenbaum. *Computer Networks*. Prentice Hall, $4^{th}$ edition, 2003.

[111] X. Tang and J. Xu. On replica placement for QoS-aware content distribution. In *Proceedings of the* $23^{rd}$ *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'04)*, Hong Kong, China, March 2004.

[112] R. Tarjan. *Data Structures and Network Algorithms*. SIAM Series in Applied Mathematics 44, 1983.

[113] A. Tsiolis and M. K. Vernon. Group-guaranteed channel capacity in multimedia storage servers. In *Proceedings of the 1997 ACM SIGMETRICS International Conference on Measurement and Modeling of Computing Systems (SIGMETRICS'97)*, pages 285–297, Seattle, WA, June 1997.

[114] University of Wisconsin-Madison eTeach project, July 2004. http://eteach.engr.wisc.edu/.

[115] S. Viswanathan and T. Imielinski. Metropolitan area video-on-demand service using pyramid broadcasting. *Multimedia Systems*, 4(3):197–208, August 1996.

[116] D. Waitzman, C. Partridge, and S. Deering. Distance vector multicast routing protocol. RFC 1075, November 1988.

[117] D. Wall. *Mechanisms for Broadcast and Selective Broadcast*. Ph.D. dissertation, Stanford University, June 1980.

[118] L. Wang, V. Pai, and L. Peterson. The effectiveness of request redirection on CDN robustness. In *Proceedings of the* $5^{th}$ *Symposium on Operating Systems Design and Implementation (OSDI'02)*, pages 345–360, Boston, MA, December 2002.

[119] Waterloo Maple, July 2004. `http://www.maplesoft.com/`.

[120] B. M. Waxman. Routing of multipoint connections. *IEEE Journal on Selective Areas in Communications*, 6(9):1617–1622, December 1988.

[121] L. Wei and D. Estrin. A comparison of multicast trees and algorithms. Technical Report USC-CD-93-560, University of Southern California, September 1993.

[122] L. Wei and D. Estrin. The trade-offs of multicast trees and algorithms. In *Proceedings of the* $3^{rd}$ *International Conference on Computer Communications and Networks (ICCCN'94)*, San Francisco, CA, September 1994.

[123] J. W. Wong. Broadcast delivery. *Proceedings of the IEEE*, 76(12):1566–1577, December 1988.

[124] W.-M. R. Wong and R. R. Muntz. Providing guaranteed quality of service for interactive visualization applications. Technical Report 990052, University of California, Los Angeles, November 1999.

[125] S. Wu and T. Kameda. Lossless VBR video broadcasting considering user bandwidth limit. In *Proceedings of the 2004 IEEE International Conference on Multimedia & Expo (ICME'04)*, Taiwan, China, June 2004. To appear.

[126] Yahoo Messenger, July 2004. `http://messenger.yahoo.com/`.

[127] A. Young, J. Chen, Z. Ma, A. Krishnamurthy, L. Peterson, and R. Wang. Overlay mesh construction using interleaved spanning trees. In *Proceedings of the* $23^{rd}$ *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'04)*, Hong Kong, China, March 2004.

[128] E. W. Zegura, K. Calvert, and S. Bhattacharjee. How to model an internetwork. In *Proceedings of the* $15^{th}$ *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'96)*, pages 594–602, San Francisco, CA, April 1996.

[129] S. Q. Zhuang, B. Y. Zhao, A. D Joseph, R. H. Katz, and J. D. Kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *Proceedings of the* $11^{th}$ *International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'01)*, pages 11–20, Port Jefferson, NY, June 2001.