Scalable Reliable On-Demand Media Streaming Protocols

A Thesis Submitted to the College of

Graduate Studies and Research

in Partial Fulfillment of the Requirements

For the Degree of Doctor of Philosophy

in the Department of Computer Science

University of Saskatchewan

Saskatoon, Saskatchewan

by

Anirban Mahanti

© Copyright Anirban Mahanti, March 2004. All rights reserved.

Permission To Use

In presenting this thesis in partial fulfillment of the requirements for a Postgraduate 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

This thesis considers the problem of delivering streaming media, on-demand, to potentially large numbers of concurrent clients. The problem has motivated the development in prior work of scalable protocols based on multicast or broadcast. However, previous protocols do not allow clients to efficiently: 1) recover from packet loss; 2) share bandwidth fairly with competing flows; or 3) maximize the playback quality at the client for any given client reception rate characteristics.

In this work, new protocols, namely Reliable Periodic Broadcast (RPB) and Reliable Bandwidth Skimming (RBS), are developed that efficiently recover from packet loss and achieve close to the best possible server bandwidth scalability for a given set of client characteristics. To share bandwidth fairly with competing traffic such as TCP, these protocols can employ the Vegas Multicast Rate Control (VMRC) protocol proposed in this work.

The VMRC protocol exhibits TCP Vegas-like behavior. In comparison to prior rate control protocols, VMRC provides less oscillatory reception rates to clients, and operates without inducing packet loss when the bottleneck link is lightly loaded. The VMRC protocol incorporates a new technique for dynamically adjusting the TCP Vegas threshold parameters based on measured characteristics of the network. This technique implements fair sharing of network resources with other types of competing flows, including widely deployed versions of TCP such as TCP Reno. This fair sharing is not possible with the previously defined static Vegas threshold parameters.

The RPB protocol is extended to efficiently support quality adaptation. The Optimized Heterogeneous Periodic Broadcast (HPB) is designed to support a range of client reception rates and efficiently support static quality adaptation by allowing clients to work-ahead before beginning playback to receive a media file of the desired quality. A dynamic quality adaptation technique is developed and evaluated which allows clients to achieve more uniform playback quality given time-varying client reception rates.

Acknowledgments

God does not care about our mathematical difficulties.

He integrates empirically.

Albert Einstein

With respect and gratitude, I acknowledge my supervisor Dr. Derek Eager, whose expertise, guidance, support, encouragement, and patience has made this dissertation possible. His feedback on my thesis research has vastly improved the quality of this dissertation and provided an enthralling educational experience.

I would also like to thank Dr. Mary Vernon for providing feedback on my research and her assistance with writing papers for publication. Also, I must add that I truly enjoyed visiting her research group at the University of Wisconsin. She has always been a wonderful host and the trips to Wisconsin were a great learning experience.

I would also like to thank Dave Sundaram-Stukel and Jeremy Parker for their assistance with the prototyping component of my thesis.

A very special thanks goes out to Dr. Carey Williamson. Dr. Williamson co-supervisor my M.Sc. thesis and was largely responsible for convincing me to join the Ph.D. program at the University of Saskatchewan. He has been a wonderful mentor who guided me and inspired me to consider academia as a career path.

I must also acknowledge my committee members: Dr. Mostafa Ammar (external), Dr. Rick Bunt, Dr. Carl Gutwin, Dr. Dwight Makaroff, and Dr. Dave Dodds. Their comments and suggestions have improved this thesis.

The office staff members of the Department of Computer Science have been wonderful. Jan Thompson, our graduate correspondent and the unofficial “mom” of all graduate students in the department, deserves my sincerest thanks for going out of her way to make life easier.

Spending six and a half years as a graduate student can be quite daunting. Several fellow graduate students and friends have made my life colorful and as pleasant as it could possibly be. I would like to thank my friends Mushir Ahmed and Nisar

Hassan for their open invitation to have dinner at their place on more than one occasion. I would also like to thank Jayakumar Srinivasan for many useful technical discussions. A very special thanks to my friend Richard Hogan for the many entertaining debates, proof-reading my thesis and papers, and generally putting up with my frustration. The one sad part of graduating is that I am going to miss going for coffee with Richard.

Finally, I would like to make a special mention of my family. I wouldn't be completing graduate studies if they didn't teach me the value of hard work and dedication. I would also like to acknowledge my wife Bhaswati for making a *special* difference to my life.

Table of Contents

List of Tables

List of Figures

List of Acronyms

ACK: Acknowledgment

AIMD: Additive Increase, Multiplicative Decrease

ALI: Average Loss Interval

BGP: Border Gateway Protocol

CBR: Constant Bit Rate

DVMRP: Distance Vector Multicast Routing Protocol

EWMA: Exponential Weighted Moving Average

FEC: Forward Error Correction

FLID-DL: Fair Layer Increase/Decrease with Dynamic Layering

FTP: File Transfer Protocol

HMSM: Hierarchical Multicast Stream Merging

HPB: Heterogeneous Periodic Broadcast

HTTP: HyperText Transfer Protocol

IGMP: Internet Group Management Protocol

IP: Internet Protocol

LDA: Loss-Delay Based Adjustment

MBGP: Multipoint Extensions to BGP

MPEG: Motion Pictures Expert Group

MRTT: Multicast Round Trip Time

MSDP: Multicast Source Discovery Protocol

PIM: Protocol Independent Multicast

PLM: Packet pair receiver-driven Layered Multicast

RAP: Rate Adaptive Protocol

RBS: Reliable Bandwidth Skimming

RLC: Receiver-driven Layered Congestion

RLM: Receiver-driven Layered Multicast

RMRC: Reno Multicast Rate Control

RPB: Reliable Periodic Broadcast

RTP: Real-time Transport Protocol

RTSP: Real-Time Streaming Protocol

RTT: Round-Trip Time

STB: Set-Top Box

TCP: Transmission Control Protocol

TFRC: TCP Friendly Rate Control

UDP: User Datagram Protocol

VBR: Variable Bit Rate

VMRC: Vegas Multicast Rate Control

WEBRC: Wave and Equation Based Rate Control

WWSTP: Wisconsin Washington Saskatchewan Transport Protocol

WWW: World Wide Web (or simply the “Web”)

Chapter 1

Introduction

Recent years have witnessed the mushrooming of networked multimedia applications on the Internet. This growth has been fueled by advances in high speed networking, workstation performance, compression technologies, and also by the widespread availability of the World-Wide Web. The Web has successfully provided an abstraction of the Internet to the common user that enables use of complex applications without knowing the details of their functioning.

A large variety of multimedia applications are available on the Internet that include services such as movies-on-demand, audio-on-demand, news-on-demand, distance learning, interactive video games, live broadcasts, Internet telephony, and video conferencing. Particularly popular are video-on-demand applications that deliver *stored* media files *on-demand* to clients.

Two contrasting service models can be identified for video-on-demand systems. One approach is to *download* the entire media file to the client before beginning playback. This approach, however, may result in unacceptably high latency to begin playback since digitally encoded media files can be quite large, depending on the duration of the content and the compression quality. To reduce the latency to begin playback (also known as the "start-up" delay), *streaming* media servers allow clients to begin media playback after filling a buffer with an initial portion of the media file. The remainder of the media file is delivered as the video is played.

Early work on video-on-demand deployment, such as AOL Time Warner's market test in Orlando, used existing cable television networks [101]. Subscribers were provided with a specialized hardware device called a set-top box that acted as an interface between the television set and the network, and could be used to browse content at the server, make requests for media files, and control the playback of media files. Most of the early work on commercial video-on-demand deployment met with only limited success. First, stiff competition was offered by traditional services such as video rentals and pay-per-view. Second, providing quality similar to, or better than, these traditional services in a cost effective manner was challenging. Third, major Hollywood studios were quite skeptical about retailing movies directly to the consumers. More recently, realizing the potential of profitably providing media distribution to homes, major Hollywood studios have formed alliances with cable, phone, and computer companies to provide video-on-demand services over cable networks and the Internet. Current trends indicate increasing availability of commercial video-on-demand services from cable/satellite television service providers, and to some extent, on the Internet as well.

This thesis considers the problem of streaming popular media files to clients distributed across a network. A key challenge is to design protocols that are "scalable", in the sense of being able to efficiently serve large numbers of concurrent clients, each with very low start-up delay. Also, in some environments, effective packet loss recovery is required, as well as adaptation to the available bandwidth[1] on the path from the server. This thesis proposes protocols that address these problems, and evaluates their performance.

The remainder of this chapter is organized as follows. Section 1.1 describes the research objectives. The primary contributions of the thesis are enumerated in Section 1.2, followed by a thesis roadmap in Section 1.3.

[1] Throughout this thesis, the term "bandwidth" refers to the amount of data transmitted in a fixed amount of time, and does *not* imply a band of frequencies or wavelengths.

1.1 Problem Definition and Motivation

Video consists of a sequence of frames that convey their meaning only when the timing relationships between the frames are maintained during playback. Therefore, a streaming media server must reserve sufficient storage and network I/O bandwidth for all data to be delivered in time for playout at the client. The unit of server capacity or "bandwidth" associated with a single data stream is referred to as a *channel*. Since a large number of (potentially long duration) streams might be in progress at a given point in time, server channels must be carefully managed.

In the conventional approach to media-on-demand, a separate server channel is allocated for each client request. With this approach, the required server bandwidth grows linearly with the client request rate. One approach to reducing the demands on the server and network bandwidth is to cache (pull strategy) or replicate (push strategy) media files (or portions thereof) at sites closer to the requesting clients [130, 116, 7]. A complementary strategy is to make use of *multicast* delivery, with which many clients can be served using a single stream. Multicast can be an effective approach owing to the typical skewness in the distribution of client requests across media files. Studies of video rental patterns [32] as well as recent workload characterizations of streaming media servers [28, 8] suggests that the top 10-20% of the popular media files may account for 60-80% of all client requests. The design of improved *scalable on-demand streaming* protocols based on multicast is an objective of this thesis work.

On-demand streaming applications differ from *live* (e.g., Internet "webcast" of a sporting event) or *scheduled broadcast* (e.g., Internet radio) applications, in which the server's transmission schedule is independent of client arrivals. That is, in contrast to on-demand streaming, clients here have no control of the programming and simply "tune in" as desired to receive the program currently being multicast. Multicast streaming is much more complicated for on-demand applications, as different client requests for the same portion of a media file may arrive at different times.

The challenges posed by the heterogeneity of the Internet, in particular differing available bandwidths and packet loss rates on the paths to different clients, are common to live and scheduled broadcast as well as on-demand streaming media applications. In the Internet TCP/IP architecture, two transport layer protocols are available, namely the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP) [71, 120]. TCP is a reliable data transfer protocol and implements congestion control schemes that adapt the sending rate of the application in accordance with changing network conditions, and is used by a majority of the applications on the Internet. UDP is an unreliable data transfer protocol that incorporates no packet loss recovery or congestion control schemes of its own, and thus applications can use their own schemes tailored to their own needs and environments. Since TCP employs retransmissions and congestion control schemes that yield variable transmission rates and packet delays, many media streaming systems use UDP as the transport protocol.

Streaming media applications that employ UDP require mechanisms to deal with packet loss. Packet loss can occur for the following reasons: 1) congestion in the network can lead to router buffers filling and thus packets being dropped; 2) queuing delays in the network can result in packets arriving later than their scheduled playout time; and 3) packets may be corrupted. The impact of packet loss on the quality of the delivered stream depends on the characteristics of the loss process, and the media compression algorithm. With compression algorithms such as those used in MPEG-2, a single packet loss can degrade media quality over several frames, if the lost packet contained a portion of an I-frame [53]. Measurements of MPEG streaming on the Internet show that packet loss rates as low as 3% can translate into frame error rates as high as 30%, thus underscoring the necessity of error control mechanisms [18].

At low loss rates, simple error concealment techniques such as use of the previous video frame in place of a partially received frame, or interpolation from close by video frames are sufficient [100]. At higher loss rates, more complex schemes that employ a mix of sender and receiver based mechanisms are required. For example, parity packets may be set along with the data packets to allow recovery of lost packets at the receivers [100]. However, such solutions have only been developed for unicast streaming, or multicast streaming of live or scheduled broadcasts [135, 49, 26, 92, 17, 75, 122, 30]. An objective of this thesis is to develop solutions that are both scalable and reliable for on-demand media streaming.

In addition to packet loss recovery, congestion control schemes are required to prevent starvation of flows, ensure inter-protocol fairness, and increase network utilization [44]. It is widely agreed that the current performance of the Internet largely depends on the flow and congestion control capabilities built into TCP, which is responsible for the majority of the traffic. Therefore, it is recommended that multimedia flows that use UDP be TCP-compatible (or "TCP-friendly") such that their bandwidth usage does not (often or on long time scales) exceed that of a conformant TCP flow under similar conditions [19]. The bandwidth usage of a streaming media flow is typically altered by modifying the amount of data sent (implying a change in the playback quality), in contrast to conventional TCP applications in which the same data is sent, but slower.

The design of such rate control schemes is complicated by the conflicting goals of promptly reacting to changes in the network conditions and of making only infrequent changes in the client perceived media quality. These issues have been addressed to some extent in the context of unicast multimedia rate control [104, 45]. Rate control protocols have also been proposed in the multicast context, but their operation is complicated by the substantial heterogeneity in client transmission paths with respect to delay, bandwidth, and loss characteristics.

One method for accommodating heterogeneous transmission rates to clients is to use *multirate* congestion control schemes [88, 125, 75, 107, 22, 74, 119]. These schemes assume a layered encoding of the media file into a base layer and a small number of enhancement layers, and that each layer is multicast on a separate channel. The base layer by itself yields a minimal quality of playback at the client. Quality is improved by receiving one or more enhancement layers in addition to the base layer. Depending on the available bandwidth on the transmission path to a client, that client can decide how many layers to receive. Another objective of this thesis is to develop improved multirate congestion control algorithms.

The available bandwidth to clients of a streaming media server may differ substantially across the client population of a streaming server (because of network access limitations for example). Streaming protocols proposed in the literature are designed assuming homogeneous client reception capabilities, and therefore, a low bandwidth client is limited to receiving a comparatively lower quality media than a client with higher bandwidth, or incur substantial increase in the latency to begin playback if higher quality playback is desired. Furthermore, available bandwidth to clients will be time-varying owing to use of congestion control protocols. Varying the playback quality in direct response to changes in the available bandwidth predicted by a congestion control algorithm may result in frequent changes in the playback quality [104]. A final objective of this thesis is to support flexible *quality adaptation* mechanisms such that the media stream received by a client in response to the available bandwidth on the path from the server allows media playback quality that is as high as that permitted by the average available bandwidth.

Putting together the above objectives, the primary goal of this research is to develop efficient, scalable, reliable, on-demand streaming protocols. The characteristics desired in such streaming protocols are:

- **Scalable**: The protocols should be able to serve a large number of client requests for a media file with low server and network bandwidth usage. In particular, the server and network bandwidth requirements should scale sublinearly with the increase in client requests for the media file.

- **Reliable**: Clients should be able to recover from packet loss as long as the loss rate is below a tunable threshold, without requiring any control feedback to the server.

- **Rate Adaptive**: The streaming protocols should adapt their streaming rate in accordance to the changes in the network conditions. Moreover, it is desirable that such rate adaptation be "TCP-friendly".

- **Quality Adaptive**: The protocols must elegantly handle the wide ranging heterogeneity of the client population, both with respect to the client network access bandwidth (i.e., as reflected by the last-mile bandwidth limitations such as imposed by dialup modems) and the transmission path characteristics (e.g., loss rates, or congestion controlled rates). In general, each client should be able to receive a media quality of its choice with minimum possible start-up delay. Further, the playback quality should be relatively uniform over time, and remain unaffected by short-term oscillations in available bandwidth.

1.2 Contributions

The main contributions of this thesis are as follows:

- Development of new scalable on-demand streaming protocols that efficiently incorporate packet loss recovery;
- Design of effective "TCP-friendly" rate adaptation mechanisms that can be used with the proposed scalable streaming protocols; and
- Design of efficient quality adaptation mechanisms applicable with the above protocols and mechanisms.

These contributions are elaborated on in the following sections.

1.2.1 Packet Loss Recovery

New periodic broadcast protocols, namely Reliable Periodic Broadcast (RPB) and Reliable Bandwidth Skimming (RBS), are developed that provide scalable on-demand streaming with packet loss recovery. These protocols can recover lost packets provided the loss rate is below a tunable threshold, can efficiently handle bursty loss, and exhibit fruitful tradeoffs among start-up delay, loss rate, and achievable client data rate.

The RPB protocols also improve the state-of-the-art in the design of *periodic broadcast* protocols [127, 2, 62, 69, 54, 95, 59]. Periodic broadcast protocols divide a media file into segments that are periodically multicast according to a predetermined transmission schedule. Clients requesting a media file receive a reception schedule informing them of when to tune in to receive the media segments, along with a start-up delay for beginning playback. Typically, these protocols require clients to be able to receive data on multiple channels concurrently. Prior work on periodic broadcast protocols assumes that the aggregate achievable transmission rate to each client is at least twice the media playback bit rate. The RPB protocol family has the favorable property of allowing scalable and efficient media delivery even when the aggregate achievable transmission rate to a client is only a small percentage (e.g., 25%) greater than the media playback bit rate. Thus, these protocols allow higher media quality (and thus a higher media playback bit rate) to be used. The RPB protocols also efficiently handle bursty losses.

The RBS protocol extends bandwidth skimming protocols to support packet loss recovery [41]. Bandwidth skimming protocols initiate a new multicast stream in response to each request for a media file. Streams delivering the same media files can be dynamically "merged" using a portion of the available bandwidth at each client. An implementation of the RBS protocol is presented as "proof of concept" of the proposed approach.

Further, new scalability bounds are developed for both immediate service protocols such as bandwidth skimming as well as periodic broadcast protocols, assuming alternative packet loss recovery techniques. The results suggest that the new protocols can achieve close to the best possible server bandwidth scalability for a given set of client characteristics.

1.2.2 Rate Control

A new equation-based multi-rate congestion control protocol, called Vegas Multicast Rate Control (VMRC) is developed. The protocol is based on a recently proposed throughput model for TCP Vegas, a variant of TCP that uses queuing delay as an indication of congestion [112]. In VMRC, each client independently applies the model to determine an appropriate reception rate to the receivers. The VMRC protocol addresses shortcomings of the multirate congestion control protocols proposed in the literature [22, 74, 75, 76, 88, 107, 119, 122, 125]. The VMRC protocol is "TCP-friendly" and like TCP Vegas, can operate without introducing packet losses while probing the network for additional available bandwidth. The VMRC protocol allows receivers behind a common bottleneck link to converge quickly to the same subscription level, without using any explicit synchronization policy. A detailed performance evaluation of the VMRC protocol is carried out to justify the main protocol design choices. Such a study has not been carried out for any other multicast congestion control protocol.

The VMRC protocol introduces a technique for dynamically setting the values of the TCP Vegas threshold parameters. TCP Vegas, as originally proposed, makes a static choice of the threshold parameters. Depending on the chosen values, the Vegas flow can be either aggressive or conservative with respect to competing traffic. The dynamic threshold estimation technique proposed here adjusts the threshold parameters based on the estimates of loss rate, round-trip time, and queuing delay, and allows the flow to compete more fairly. This technique could be incorporated in TCP Vegas implementations, and may potentially aid in incremental deployment of TCP Vegas in the Internet.

1.2.3 Quality Adaptation

The RPB protocols are developed assuming homogeneous client reception rates. An extension of RPB termed Heterogeneous Periodic Broadcast (HPB) is developed that provides better support for heterogeneous clients than existing periodic broadcast protocols. The new HPB are optimized for a range of client reception rates and allow efficient tradeoffs between start-up delay and media quality.

Previous work on multicast in the context of streaming media delivery assumed that the media playback quality changes in direct response to changes in the available bandwidth to the client (see [88] for example). In the context of unicast streaming, work-ahead during playback has been used to smooth short-term variations in available bandwidth. In this work, an effective work-ahead based adaptation mechanism is developed and a performance study of this scheme demonstrates that relatively smooth playback can be achieved at the clients.

1.3 Thesis Roadmap

The remainder of the thesis is organized as follows. Chapter 2 reviews the prior work on multimedia networking that is most relevant to this work. Chapter 3 develops the new periodic broadcast and bandwidth skimming protocols that can recover from packet loss, namely Reliable Periodic Broadcast (RPB) and Reliable Bandwidth Skimming (RBS), respectively. Chapter 4 proposes a new multirate congestion control protocol called Vegas Multicast Rate Control (VMRC) that can be used in the context of scalable on-demand streaming. Chapter 5 describes quality adaptation techniques for use with scalable streaming protocols. Chapter 6 presents conclusions and directions for future work.

Chapter 2

Background

Pay-per-view services available on cable television require viewers to tune in to a program at a predetermined time. Video-on-demand aims to further this concept by storing digital content and allowing access to any content at any time. With advances in recent years in computing capabilities, digitization and compression techniques, high bandwidth storage subsystems, and communication infrastructure, video-on-demand services have become a reality in cable/satellite television networks, corporate networks, and even on the Internet.

Several digital cable service providers are offering access to a broad range of programming content to their customers. A digital set-top box (STB) acts as an interface between the television set at the customer's premise and the cable network. Using the STB, a customer can select programming of their choice from a library of titles and start viewing the program on their television at a time of their choosing with full VCR functionality including pause, rewind, and fast forward. Once a program has been ordered, the customer can start watching it immediately, or download it to watch at a later time.

Video-on-demand applications have also been widely deployed in corporate and private networks. For example, many universities are using these services for their online classes. On the Internet, movie trailers, music videos, news clips etc. are available for streaming. More recently, several companies have started streaming full length movies on-demand to clients with broadband Internet connections. These developments have been facilitated by a wide body of research on video-on-demand systems. Rather then attempting to cover the entire spectrum of video-on-demand research, this chapter will discuss related work that is most relevant to this thesis.

The remainder of this chapter is organized as follows. Section 2.1 presents an overview of stored media file streaming. Section 2.2 briefly discusses the multicast delivery model assumed by the scalable delivery techniques reviewed in Section 2.3. Issues pertaining to congestion control and error control of multimedia flows are discussed in Sections 2.4 and 2.5, respectively.

2.1 Architectures for Streaming Stored Media Files

One approach to streaming is to store the compressed media files on a Web server and deliver them using the protocol used for transferring Web pages, namely the Hypertext Transfer Protocol (HTTP)[1]. A client can select the desired media file by clicking on a hyperlink identifying the media file. The client's browser sends an HTTP GET request for the file to the server, and the server responds by delivering the contents of the media file in the body of an HTTP response message. On receipt of the response message from the server, the browser launches the appropriate media player by using the information available from the *Content-Type* field in the header of the response message, and passes the contents of the file to the media player. Although this approach is simple, it suffers from the drawback that the media file contents must be completely *downloaded* by the browser before being forwarded to the media player, resulting in unacceptable delays in beginning playback for large media files.

An alternative to the above download approach is *progressive download*, where the media file is directly sent to the media player, instead of the media file transiting via the browser. To enable direct communication between the server and the media player, the content provider must create, for each media file, a *meta file* that contains

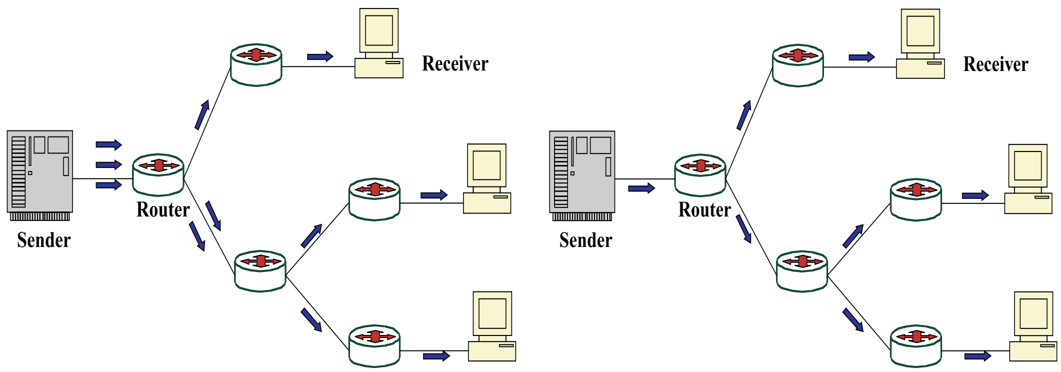
[1]HTTP is an application-level protocol that employs the Transmission Control Protocol of the Internet Protocol suite to provide reliable bidirectional communication channels between the server and the client.

information regarding the content type and the hyperlink to the actual media file. Clients select a media file of their choice by clicking on a hyperlink identifying the desired media file; however, the hyperlink is now for the meta file and not the actual contents.

The meta file is downloaded to the browser, and after examining the contents of this metafile, the browser launches the appropriate media player and passes the contents of the meta file to the player. The media player determines that the file passed to it by the browser is a meta file, reads the URL of the original media file, and initiates an HTTP GET request for the media file. The server delivers the contents of the media file using an HTTP response message. Thus, playback can begin after downloading some predetermined amount of data.

The main disadvantage of this approach emanates from the use of HTTP as the protocol for delivering the media file. The HTTP protocol uses TCP as the underlying protocol for providing reliable delivery. TCP results in highly variable transmission rates and packet delays. Therefore, large buffers are necessary to enable jitter-free playback. HTTP is a connection-oriented protocol that is fundamentally non-scalable for one-to-many communication. To overcome these difficulties, often a specialized *streaming media server* is used, that can be easily configured to use protocols designed especially for streaming.

Two of the most popular streaming servers are Real Network's Real Server, and Microsoft's Windows Media Server. When a specialized server is used, the initial interaction for obtaining the meta file is as described above. However, the interaction between the media server and the media player is different. The media player uses a control protocol such as the public domain Real-Time Streaming Protocol (RTSP) [114] for initiating and controlling the delivery of the media file from the server (e.g., provide VCR-like functionality such as play, pause, rewind, and forward). Streaming servers packetize data using protocols such as the public domain Real-Time Transport Protocol (RTP) [113] to provide services such as payload type identification, sequence numbering, time stamping, and source identification. Note that RTP was proposed as a layer within the application layer, and data packetized



(a) Unicast Datagram Delivery (b) Multicast Datagram Delivery

Figure 2.1: Comparing Unicast and Multicast Communication Models

using RTP is delivered using an application-level delivery protocol, at a rate determined in part by the media encoding and the specific protocol (e.g., a proprietary protocol, HTTP, or a scalable streaming protocol that employs multicast) in use.

Throughout the above discussion, it is assumed that clients use a computer as the display unit, with requests for media files being made using a Web browser and the contents of the media file being displayed using a media player. The same functionality can also be provided by a STB attached to a conventional television set.

2.2 Multicast Service Model

Many multimedia applications require sending the same data to multiple clients. In this context, sending the same data using separate point-to-point unicast connections results in much wastage of server and possibly also network resources, as shown in Figure 2.1(a). An attractive alternative is to use the multicast extension to the IP layer that provides scalable point-to-multipoint datagram delivery for group communication. Scalability is achieved by sending a single datagram from the sender, that is replicated by the routers whenever the datagram must be forwarded on multiple links, as illustrated in Figure 2.1(b). Clearly, multicast service requires both the end systems and the network to be multicast capable.

In the traditional IP-multicast service model [33], a set of clients form a logical entity called a *group*. All members of a group can be identified with a single class-D multicast address. There is no restriction on the size of a group or its location, and any sender (even those that are not members of the group) can send to all members of a group by addressing data packets to this multicast address. The data packets are delivered using IP-style semantics, with no guarantees of in-order or loss-free delivery, although these features may be provided at the application layer. The model puts the onus on group members to inform their nearest router of the intent to receive packets sent to a multicast address. This functionality is provided by the Internet Group Management Protocol (IGMP) [43] operating between clients on the subnet and their designated router.

Note that IGMP only manages clients on a subnet, and is not concerned with the multicast spanning tree used to distribute datagrams between designated routers in different subnets. Multicast routing algorithms generate spanning trees to connect routers that have clients belonging to a particular multicast group, and consist of protocols that: 1) efficiently route datagrams within a domain (*intra*domain routing); and 2) route datagrams between different domains (*inter*domain routing). The algorithms commonly in use are discussed in the next two subsections. For a detailed survey of multicast routing protocols, the reader is referred to [9, 111].

2.2.1 Intradomain Routing

The original multicast routing protocol, Distance Vector Multicast Routing Protocol (DVMRP) [128], is a *flood* and *prune* protocol. When a source first starts sending, it will flood the network by sending datagrams on all its outgoing links. In turn, the receiving routers will forward the datagram on all of their outgoing links provided the datagram arrived on the shortest reverse path back to the source. DVMRP maintains its own routing tables and packets not received on the reverse shortest path are dropped. In this approach, many parts of the network will receive traffic although no group members are present. Therefore, routers with no attached group members send a prune message to the upstream router. Each router in the network employs this flood and prune protocol until the distribution tree is established. Pruned branches periodically timeout (default value is two hours), and datagrams reflood the network until unwanted branches are pruned back. Graft messages are used to add back pruned branches to the multicast tree.

This protocol works well when group members are located on most subnets, but is inefficient when group members are sparsely distributed. A key scalability issue is that every router in the network maintains state for all (source, group) pairs, regardless of whether any group members exist downstream. Another problem is picking a suitable timeout period for pruned branches. Clearly, there exists a tradeoff between the join latency and the frequency of flooding. In the sparsely distributed scenario, a direct consequence of ensuring quick joins is an increased frequency of flooding routers with unnecessary data. The increased use of multicast in wide-area networks led to the development of protocols that build a shortest path tree from a designate point in the network, referred to as a *core* [12] or *rendezvous point* [35], which is shared between all sources.

The Protocol Independent Multicast (PIM) [35, 34] architecture is the commonly deployed routing algorithm in the Internet, and was proposed with the following objectives: 1) the protocol should be able to use the available unicast routing tables to generate the multicast trees; 2) avoid the overhead of broadcast and prune protocols when group members are sparsely distributed over the Internet; and 3) support good quality distribution trees for heterogeneous receivers by supporting both group shared and source specific trees. PIM provides two modes of operation, namely a *dense* mode and a *sparse* mode. As the names suggest, the former is useful when group members are densely distributed, while the latter is useful when the group members are sparsely distributed in the network. The operation of the protocol in the dense mode is similar in spirit to that of DVMRP. For details, the reader is referred to [34].

When operating in sparse mode, receivers send join requests to a known core or *rendezvous point* (RP). Intermediate routers process this request, setting up a shared tree from the RP for the group. Each source sends data to the RP, and the packets are multicast on the shared tree. Thus, the RP acts as a "meeting point" for sources and receivers. A novel feature of this protocol is the ability to switch from group shared trees to source specific trees, depending on the volume of traffic generated by the source(s). For example, if receivers experience intolerable end-to-end delays, it may decide to switch to a more optimal source specific tree.

The main drawbacks of PIM are related to the use of an RP. Specifically, the RP is a single point of failure, and group shared trees are often non-optimal.

2.2.2 Interdomain Routing

This section briefly discusses the current approach to multicast routing across domains. Since PIM-SM is the most widely deployed multicast routing protocol, it is necessary that all interdomain multicast routing protocols interoperate with PIM-SM.

Unicast interdomain routing is supported using the Border Gateway Protocol (BGP) [106]. Since PIM-SM is protocol independent and uses any available routing protocol for reverse path lookup, BGP can be used for the reverse path lookup part of interdomain multicasting. However, it may be desired that unicast and multicast traffic utilize different links at border routers. The extensions to BGP that allow carrying multicast routes are called Multiprotocol Extensions to BGP (MBGP) [15].

Note that MBGP only provides the capability of determining the next hop to a host, but does not exchange RP information across domains (Specifically, does not inform RP in one domain about sources in another domain). This functionality is provided by the Multicast Source Discovery Protocol (MSDP) protocol, and has been developed for dynamic interdomain source to RP mapping [9]. MSDP uses TCP to reliably exchange source information between domains.

2.2.3 Alternative Service Model

The current multicast architecture suffers from a number of drawbacks in areas such as scalability, network management, security, and multicast address allocation

[36]. These drawbacks have been a serious impediment to widespread multicast deployment, and have motivated research in alternative service models [58, 31, 65].

Many complications of the traditional IP-multicast architecture can be avoided using the single source approach, as outlined in the EXPRESS model [58]. In this model, address management is simplified by defining a *channel* as a tuple consisting of the source IP address and a destination channel number (a 24 bit value allowing 2^{24} channels). This modification provides many more multicast channels than available in the original IP multicast model and has the added advantage of not requiring any global address coordination. Interested receivers subscribe to a channel by sending a request to the network explicitly specifying both the source IP address and the channel number. Furthermore, a source can control other clients' ability to subscribe to channels, and distribution trees can be shortest paths back to the source. There is no need for flood and prune type algorithms or rendezvous point based algorithms. Also, a simpler charging model is possible because there is one owner per channel which is easily identifiable; ISPs may charge the source based on the number of channels it uses. Multisource applications may be supported by allocating one channel at each source. Alternatively, the application may designate a coordinator node, referred to as the *relay* node in [58], that coordinates access of the secondary sources to the main channel.

Many recent proposals have argued for application-level multicast service, rather than providing multicast as a network primitive [31, 65]. The basic approach is to construct a virtual network, called an *overlay*, that connects participating nodes using unicast paths, as illustrated in Figure 2.2. Non-leaf nodes in the overlay can be either dedicated machines [65], or simply participating clients [31]. Application-level multicast poses several challenges. First, an algorithm to construct efficient overlay structures that minimizes end-to-end delay, total network resource usage, and duplicate transmission of packets on network links is non-trivial to design. Second, the overlay should be able to adapt to the dynamics of the participating nodes. For example, the overlay structure must remain connected in case of a node failure, and should quickly adapt to addition/removal of overlay nodes. Third, the overlay

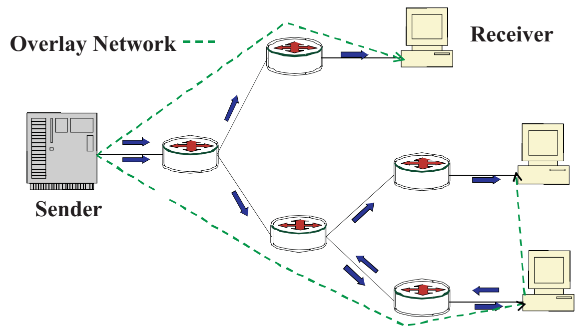


Figure 2.2: Example of Application-Level Multicast

should also adapt to changes in the underlying substrate network (e.g., pick a better overlay path when available). Generally, mechanisms should be implemented that adaptively improve an overlay over time.

2.3 Scalable Streaming Protocols

The basic idea behind scalable streaming is to enable aggregation of client requests, wherein a large number of client requests for a media file are served using a few multicast streams. This approach originated from work in the field of scalable download of relatively small sized objects using broadcast/multicast. In this scalable download context, the objective is to determine the transmission schedule of objects such that the average client delay is minimized [37, 10, 133].

In recent years, a wide array of scalable streaming protocols have been proposed, that can be broadly classified as "server push" protocols or "client pull" protocols. The "server push" techniques dedicate a fixed number of server channels per media object. Typically, these protocols divide a media object into K segments with relative lengths $l_1, l_2, ..., l_K$. Each segment is periodically multicast according to a fixed schedule. Clients often receive multiple segments concurrently at an aggregate rate that exceeds the media playback rate, and buffers data that is received ahead of when it is needed for playback. A client selecting an object obtains a schedule for tuning into the various channels, and a latency to begin playback that allows receiving all media data by the time it is needed. Because of the periodic nature

Table 2.1: Notation for Scalable Streaming Protocols and Scalability Bounds

Symbol	**Definition**
K	total number of segments
l_k	playback duration of the k^{th} segment (relative to the length of segment 1)
r	channel transmission rate (in units of the media playback rate)
b	aggregate client reception rate (in units of the media playback rate)
d	maximum start-up delay for media object playback
B	required server bandwidth (in units of the media playback bit rate)
T	media object playback duration
M	number of media objects delivered using the protocol
λ	average client request rate for a media object
N	average number of requests for the media object that arrive during a period of length T ($N = \lambda T$)

of multicasting the media files, these protocols are also known as *periodic broadcast* protocols. Previous work on periodic broadcasts is reviewed in Section 2.3.1.

The "client pull" protocols initiate media streams only in response to client requests. The simplest form of "client pull" scalable streaming protocol is *batching*, in which requests for the same media file that are within a small time interval are queued, and served using a single multicast stream. Various batching policies have been proposed that differ in how queued requests are scheduled [32, 3, 124]. Regardless of queuing policy, batching yields relatively high client delay. To reduce start-up latencies, several *immediate* service protocols have been proposed that initiate a new multicast stream in response to a client request, and dynamically expand the group of clients served by the stream to accommodate later client requests. Specific schemes in this later category are reviewed in Section 2.3.3.1. The basic notation used for discussing the protocols and related scalability bounds is provided in Table 2.1.

The performance of these scalable protocols depends on the media file access characteristics. Periodic broadcast protocols may be suitable for the most popular media files since the server bandwidth requirement of these protocols is independent of the request arrival rate. The less popular files can be delivered using a protocol whose bandwidth requirement grows as a function of the client request rate.

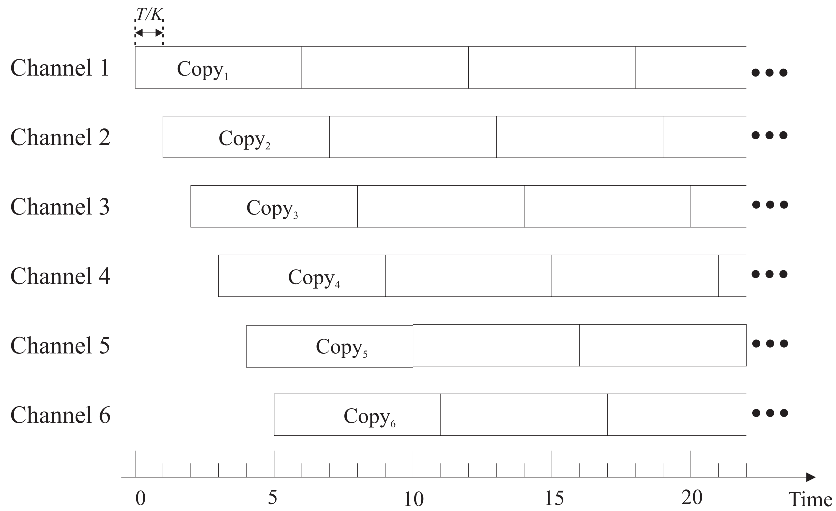


Figure 2.3: Staggered Broadcasting $(K = 6)$

2.3.1 Periodic Broadcast

2.3.1.1 Staggered Broadcasting

One of the earliest proposed scalable streaming techniques is *Staggered Broadcasting* [32]. In this protocol, entire copies of a media object are repeatedly broadcast[2] on K separate channels, each at the media playback rate. The starting times of each copy are staggered evenly across the channels by imposing a phase delay equal to T/K, as illustrated in Figure 2.3. This guarantees that any client can begin viewing the media object after a maximum start-up delay equal to T/K. With this scheme, the required server bandwidth grows linearly with decreasing start-up delay. With appropriate design of segment sizes, channel transmission rates, and segment broadcast schedules, the required server bandwidth of periodic broadcast protocols only increases *logarithmically* with decreasing start-up delay.

2.3.1.2 Pyramid Broadcasting

The idea of partitioning a media file into segments of increasing size and broadcasting these segments on separate channels was introduced by Vishwanathan and

[2]While discussing periodic broadcast protocols, the term "broadcast" will often be used instead of "multicast", to be consistent with the literature.

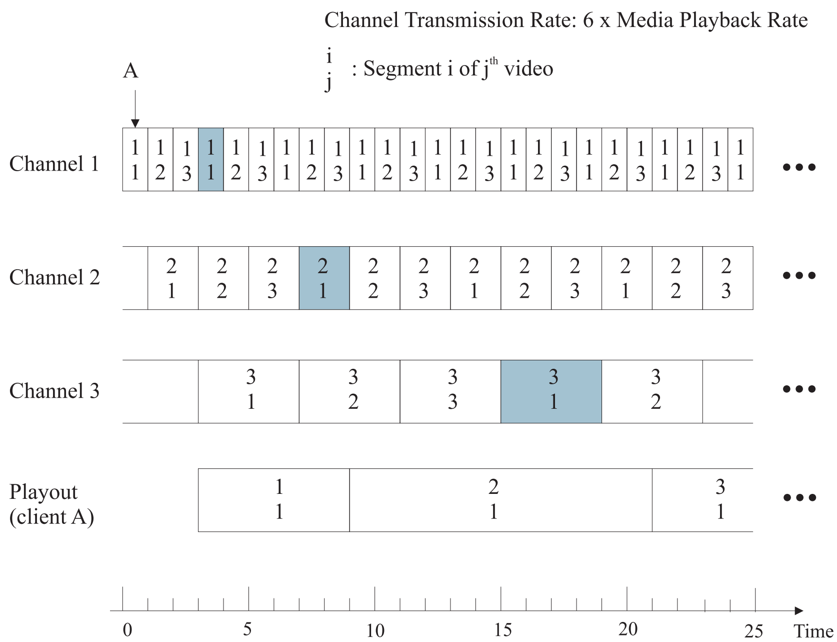


Figure 2.4: Pyramid Broadcasting $(K = 3, \alpha = 2, M = 3, r = 6)$

Imielinksi in the form of the *Pyramid Broadcasting* protocol [127]. The small first segment permits low start-up latencies while the larger later segments keep the total number of channels needed for the broadcast small. Each media object is partitioned into K segments that increase in size geometrically by a constant factor α (i.e., $l_{k+1} = \alpha l_k$). Segment k broadcasts for each of the M media objects delivered using the protocol are interleaved and broadcast on a single channel at transmission rate r. This is illustrated for $K = 3$, $M = 3$, $\alpha = 2$, and $r = 6$ in Figure 2.4. A client begins downloading the first segment of the selected media file at the first occurrence of the segment. Each segment is obtained as soon as possible after beginning to play the current segment, listening to at most two channels simultaneously. Figure 2.4 illustrates the segment reception schedule (the shaded portions), and the corresponding segment playout for the client arrival marked by the arrow.

To preserve continuity in playback, the protocol requires that the playback time of segment k should be greater than or equal to the worst case time to begin downloading segment $k+1$. Note that the upper bound on the access time for any segment occurs when the client just misses the beginning of the broadcast for the required media object, thus having to wait for the broadcast of all the M media objects to finish. Thus, continuity in playback can be achieved when

$$l_k \geq \frac{l_{k+1} \times M}{r}. \tag{2.1}$$

Since $l_{k+1} = \alpha l_k$, the channel transmission rate r is chosen to be at least αM.

The maximum start-up delay can be derived by assuming that a client tunes in immediately after the beginning of a broadcast of segment 1 of the requested media file. That is, for a server bandwidth of αMK, the maximum start-up delay is

$$d = \frac{Ml_1}{r \sum l_k} T = \frac{T(\alpha - 1)}{\alpha(\alpha^K - 1)}. \tag{2.2}$$

The value of the parameter α that results in the best start-up delay for a given server bandwidth can be determined by applying simple search techniques (α is close to e).

The Pyramid Broadcasting protocol makes more efficient use of server bandwidth than the Staggered Broadcasting protocol. For example, with the Staggered Broadcasting approach, a two hour movie would require server bandwidth equal to 60 times the playback rate to guarantee a maximum client waiting time of 5 minutes. The Pyramid Broadcasting protocol can guarantee a start-up delay under 2 minutes by using server bandwidth equal to 10 times the media playback rate ($\alpha = 2$, $M = 1$, $K = 5$). In general, unlike the Staggered Broadcasting approach, the protocol provides exponential decrease in start-up delay with linear increase in server bandwidth.

The main drawback of this protocol is the high client reception bandwidth requirement. Clients must be capable of receiving on two channels, each with bandwidth αM times higher than the media playback rate. Also, considerable client side storage space, sometimes as high as 80% of the media object size, is required to buffer preloaded segments. Finally, this protocol requires relatively high server bandwidth to achieve a given start-up delay in comparison to other techniques described next.

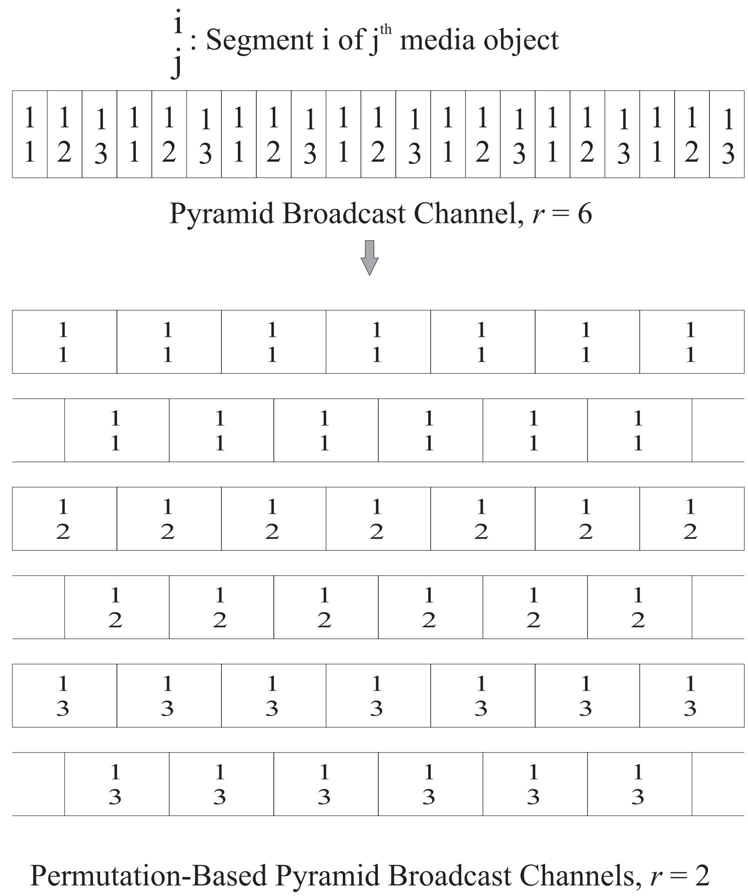


Figure 2.5: Permutation-Based Pyramid Broadcasting ($M = 3,\ P = 2$)

2.3.1.3 Permutation-Based Pyramid Broadcasting

The *Permutation-Based Pyramid Broadcasting* protocol reduces client bandwidth and storage requirements by logically separating the transmission of different media objects [1]. Note that the high client reception bandwidth required by the Pyramid Broadcast approach occurs due to the multiplexing of segments from different media objects on a single logical channel. The approach adopted in this protocol is to use a large number of low bandwidth channels in place of fewer high bandwidth channels.

As in the Pyramid scheme, each media object is partitioned into K segments that increase in size geometrically by a constant factor α (i.e., $l_{k+1} = \alpha l_k$). Each Pyramid Broadcast channel is further partitioned into $P \times M$ channels as illustrated in Figure 2.5, with $P \geq 1$ being a parameter of the protocol. Thus, segment k broadcasts for each of the M media objects can be replicated and repeatedly broadcast on P channels. The starting times of the replicated segments are uniformly staggered across these P channels, using the Staggered Broadcasting technique. For any selected media file, the client subscribes to the first available segment 1 broadcast, downloading and playing it concurrently. All subsequent segments are downloaded in sequential order. Unlike the Pyramid Broadcasting protocol, however, clients only listen to a *single* channel at a time.

The condition for jitter-free playback can be derived as follows. Since each segment is broadcast using the Staggered Broadcasting approach, the maximum delay to begin retrieval of segment $k+1$ is $\frac{l_{k+1}}{Pr}$. For any selected media object, the condition for jitter-free playback is that the beginning of segment $k+1$ must be available before the end of consumption of the current segment k, for all $1 \leq k < K$. Since clients only listen to a single channel at a time, the sum of the time to download segment k and the maximum possible latency to begin retrieval of segment $k+1$ must be less than or equal to the playback time of segment k, that is,

$$\frac{l_k}{r} + \frac{l_{k+1}}{Pr} \leq l_k. \tag{2.3}$$

Therefore, the channel transmission rate r is set to $(1+\alpha/P)$. The server bandwidth requirement is $MK(P+\alpha)$, and the maximum start-up delay is

$$d = \frac{l_1}{rP\sum l_k}T = \frac{T(\alpha-1)}{(P+\alpha)(\alpha^K-1)}. \tag{2.4}$$

This protocol reduces the maximum client side storage requirement to 50% of the media object size. However, determining appropriate parameters of this protocol appears to be difficult, and the comparative server bandwidth reduction is quite modest, in comparison to Pyramid Broadcasting.

2.3.1.4 Skyscraper Broadcasting

The *Skyscraper Broadcasting* protocol divides a media object into K segments with relative sizes given by the following recursive function [62]:

$$l_k = \begin{cases} 1 & k = 1 \\ 2 & k = 2,3 \\ \left(2 + 2\left\lfloor \frac{k}{2} \right\rfloor - k\right) l_{k-1} + \left(1 + 2\left\lfloor \frac{k}{2} \right\rfloor - k\right)\left(1 + \left\lfloor \frac{k - 4\left\lfloor \frac{k}{4} \right\rfloor}{2} \right\rfloor\right) & k \geq 4. \end{cases} \tag{2.5}$$

This function yields the following series $1, 2, 2, 5, 5, 12, 12, 25, 25, \cdots$. Each segment is repeatedly broadcast at the media playback rate on its own channel, as illustrated in Figure 2.6 for $K = 6$. A client requesting the object obtains a reception schedule for tuning into each channel to receive each of the segments, starting at the beginning of the next segment 1 broadcast on channel 1. For example, client A in Figure 2.6 obtains a schedule that includes the first shaded segment transmission on each channel. A client commences playback as soon as it begins to receive segment 1. The segment transmission schedule ensures that whichever segment 1 broadcast a client receives, the client can receive each of the remaining media segments at or before the time it is needed for playback. Since segments are increasing in size, clients that initially listen to different segment 1 broadcasts may listen to the same broadcasts of later segments. For example, any later client that is assigned one of the fully shaded segment 1 broadcasts will share the segment 6 broadcast with client A. As illustrated in Figure 2.6, greater overlap in segment reception occurs for client requests that are close together in time.

The server bandwidth requirement of the protocol is K and the maximum start-up delay is $\frac{l_1 T}{\sum l_k}$. The Skyscraper protocol represents significant a advancement over the previously discussed Pyramid protocols. First, clients are required to concurrently listen to at most two playback rate channels. Second, unlike the Pyramid protocols in which clients obtain a large part of the media object before it is required for playback, the Skyscraper protocol attempts to delay the delivery of media seg-

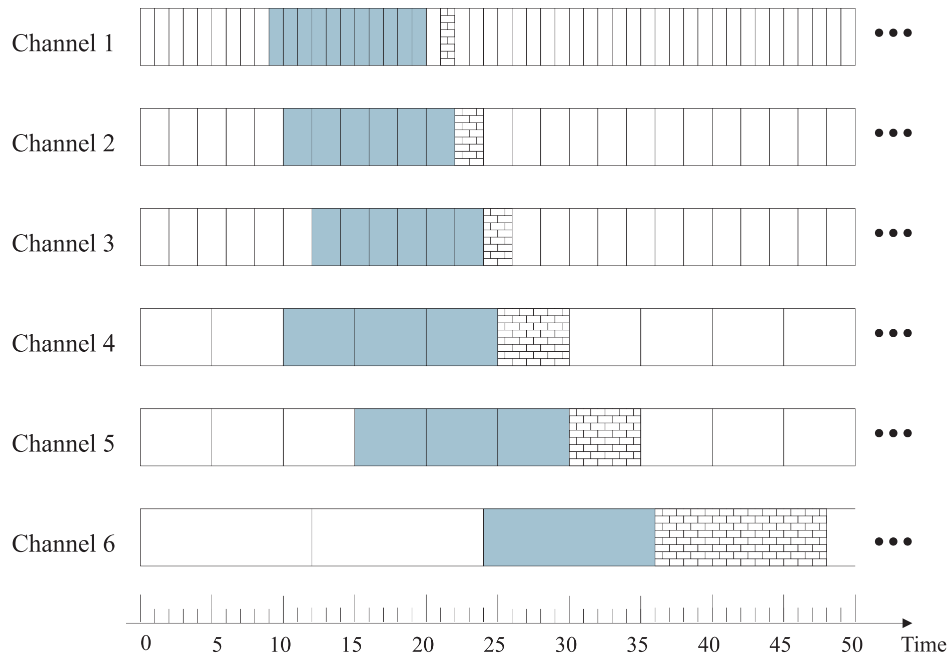


Figure 2.6: Skyscraper Broadcasting and Dynamic Skyscraper ($K = 6$)

ments until they are required for playback, resulting in better utilization of server bandwidth than the protocols discussed earlier. Third, the client buffer space requirement is somewhat lower. Specifically, clients must have buffer space equal to the size of the K^{th} segment, which is often about 40% of the media object size.

The *Dynamic Skyscraper* technique supports dynamically changing the media file being broadcast using a given set of Skyscraper channels [38]. This technique aims to increase the performance of the Skyscraper protocol when media objects exhibit time varying popularity and also extends the Skyscraper protocol for somewhat lower client request rates. The Dynamic Skyscraper protocol allocates channels in units of a *transmission cluster*, which is defined as a union of transmission schedules that encompass the same segment K broadcast, such that no segment transmission is also received by clients receiving a different segment K broadcast. Each transmission cluster can be used to broadcast a different media object. Pending client requests are assigned to an ongoing transmission cluster, if present, or allocated the next available transmission cluster according to some request service policy (such as FIFO). As an example, the fully shaded segments in Figure 2.6 show one possible transmission cluster. The next transmission cluster starts K time units later on channel 1. The start of this cluster is shown by the brick pattern in the figure. Note that there may be broadcast periods on channel 1 that can not be used by any transmission cluster, as illustrated by the gap between the fully shaded segments and the brick shaded segment on channel 1. Optimizations such as segment size progressions that do not have any holes between transmission clusters are discussed in [38].

2.3.1.5 Harmonic Broadcasting

The *Harmonic Broadcasting* protocol divides a media object into K segments of equal size, and transmits the segments on separate channels of decreasing rate [69]. Each segment k can be considered to be further divided into k equal size subsegments of lengths $l_{k,1} \cdots l_{k,k}$ that are periodically broadcast on its assigned channel at rate $1/k$ times the media playback rate as illustrated in Figure 2.7. Clients receive on all channels simultaneously, starting at the beginning of the first segment 1 broadcast following the client request. For example, in Figure 2.7 client A receives the non-striped shaded subsegments and the diagonally striped subsegments. Note that clients may receive data for individual segments out of order as shown as shown in the figure by the client A reception sequence for segment 3. Furthermore, clients listening to different segment 1 broadcasts may partially listen to the same portion of a later segment broadcast. For example, clients A and B listen concurrently to the diagonally striped subsegments on channels 3 and 4. Since clients concurrently listen on all channels, the client bandwidth and server bandwidth requirements are the same, equal to $\sum_{i=1}^{K} \frac{1}{i} = H_K$, where H_K is the harmonic sum of K.

In Harmonic Broadcasting as originally described, each client starts playback as soon as its download of segment 1 begins. However, with this policy the protocol can *not* guarantee jitter-free playback [95]. As an illustration, suppose that client A in Figure 2.7 starts playback as soon as it begins to receive the shaded segment 1 broadcast, at some time t. At time $(t + T/K)$, the client is ready to consume segment 2, but has only received subsegment l_{22}. Specifically, the client needs to playback all the data from subsegment l_{21} by time $(t+1.5T/K)$, but will not receive

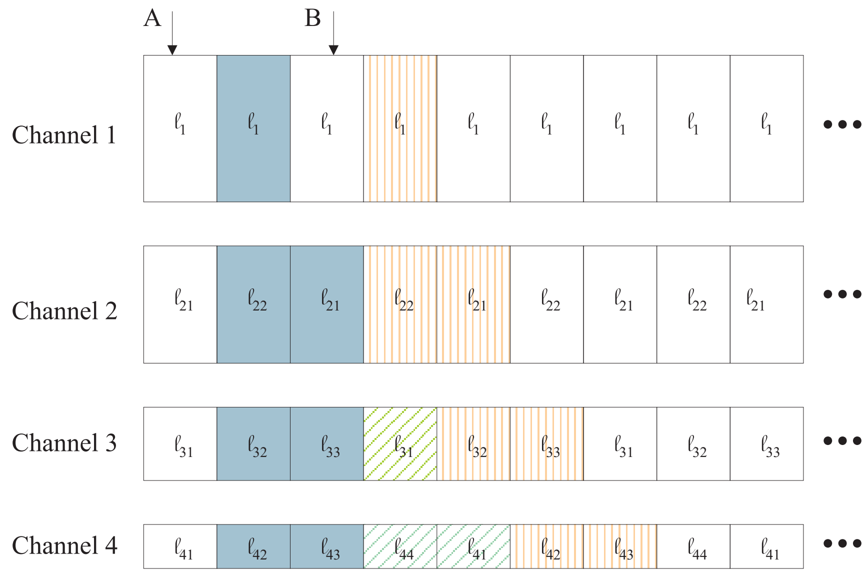


Figure 2.7: Harmonic Broadcasting ($K = 4$)

the required data until time $(t + 2T/K)$.

Paris *et al.* show that jitter-free playback can be achieved in Harmonic Broadcast systems if the clients start consuming data after completely receiving the first segment, resulting in a worst case start-up delay of $2T/K$ [95]. Two variants of Harmonic Broadcasting are proposed that achieve lower start-up delays for a fixed server bandwidth requirement than the original Harmonic Broadcasting protocol with the above modified start-up delay. A simpler and more efficient solution, however, is to start receiving data on all channels immediately after the client tunes in. The required start-up delay with this modification is deterministic, and equals the time required to download the first segment.

Note that H_K can be approximated as $\ln(K + 1) + \gamma$, where γ is the Euler constant. In the version of the protocol described above with deterministic start-up delay d, server bandwidth used is therefore approximately $\ln(T/d + 1) + \gamma$, which is within a constant factor of the lower bound for any periodic broadcasting protocol, as presented in Section 2.3.2.

The basic idea behind periodic broadcast protocols is to achieve short start-up delays by frequent broadcasts of the initial segments of the media file and low server bandwidth usage by making broadcasts of later segments infrequent. In the Pyramid-type protocols [127, 2, 62, 54], this is achieved by employing an increasing segment size progression, while the same effect is achieved by the Harmonic Broadcasting protocol and variants [69, 95] by decreasing the transmission rate of the channels that broadcast the later segments. Hybrid protocols have been proposed that combine the characteristics of the above two strategies [96]. The media file is divided into K equal-size segments and these segments are broadcast on channels at the media playback rate. The frequency of broadcasts of the later segments is decreased by using a transmission schedule in which the broadcasts of several segments are multiplexed on a single channel.

2.3.1.6 Greedy Equal Bandwidth Broadcasting

The *Greedy Equal Bandwidth Broadcasting* protocol proposed by Hu *et al.* determines the segment sizes and their corresponding transmission rates that minimize the required server bandwidth for a given client start-up delay, assuming the following: 1) a fixed number of channels each dedicated to a particular segment, are available; 2) clients receive data on all channels simultaneously; and 3) each segment is completely received prior to its playout point [60, 59]. Letting r_k denote the bandwidth allocated to periodically broadcast segment k, the relative segment size progression is defined as

$$l_k = r_k \left(1 + \sum_{j=1}^{j=k-1} l_j\right), \quad k = 1, 2, \cdots, K. \tag{2.6}$$

It is shown in [60] that the server bandwidth requirement $\sum_{k=1}^{k=K} r_k$ is minimized when channel transmission rates are equal, that is

$$r_k = r = \left(\frac{T}{d} + 1\right)^{K/2} - 1, \tag{2.7}$$

where d is the start-up latency and equals $\frac{Tl_1}{\sum l_k}$. It is interesting to note that in the limiting case (i.e., $K \to \infty$), the server bandwidth requirement of this protocol achieves the lower bound [59].

2.3.2 Scalability Bound for Periodic Broadcast Protocols

Consider the broadcast of an arbitrarily small portion of the media file dx at an offset x with respect to the beginning of the media object. For a start-up delay d, a client request arriving at time t would need to receive this portion of the media object by time $t + d + x$ to ensure continuity in playback. Therefore, if this portion of the object is broadcast at time $t + d + x$, all client requests that arrive in time $[t, t + d + x]$ can benefit from this broadcast. Clearly, no periodic broadcast of this data that is less than once every $d+x$ time units could be guaranteed to be sufficient for all possible timings of client requests. The amount of server bandwidth required to broadcast the data is $dx/(d + x)$. Therefore, the total server bandwidth B^{pb}_{min} (the superscript indicating that this is for periodic broadcasting) is [54, 16]:

$$B^{pb}_{min} = \int_0^T \frac{dx}{x + d} = \ln(\frac{T}{d} + 1). \tag{2.8}$$

The above derivation outlines an important design criteria for scalable streaming protocols: just in time delivery of media data results in the least server bandwidth usage. The periodic broadcast protocols discussed in the preceding section satisfy this objective to varying degrees. The Pyramid-type protocols broadcast the initial portions of the media file more frequently than the later portions of the media file, while the Harmonic Broadcasting protocol and its variants attempt to delay the delivery of media data by using low rate channels.

Figure 2.8 compares the required server bandwidth (measured in units of the media playback rate) as a function of start-up delay for two sample broadcasting protocols, namely Skyscraper Broadcasting and Harmonic Broadcasting. These and subsequent performance results that shows the required server bandwidth are analytically derived, unless stated otherwise. For Skyscraper Broadcasting, the required server bandwidth is plotted as a function of both the maximum start-up delay and the average start-up delay. Maximum start-up delay equals twice the average start-

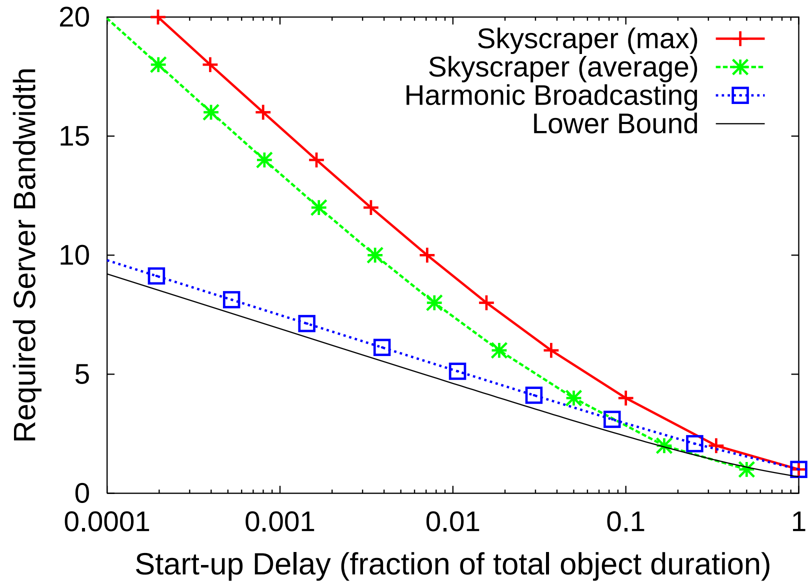


Figure 2.8: Required Server Bandwidth for Periodic Broadcast Protocols

up delay in Skyscraper systems. The figure also shows, for comparison purposes, the lower bound for any protocol that guarantees a given maximum start-up delay. Note that Harmonic Broadcasting outperforms Skyscraper in terms of the server bandwidth requirement, at the cost of requiring that clients simultaneously listen on all multicast channels broadcasting the requested media file. Skyscraper systems, however, require that clients listen to at most two server streams simultaneously. A question addressed in this thesis is whether or not new protocols can be devised that have the scalability of Harmonic Broadcasts, and yet require relatively low client reception bandwidth like the Skyscraper protocol.

2.3.3 Immediate Service Protocols

2.3.3.1 Adaptive Piggybacking

Adaptive Piggybacking "merges" streams by altering their playback rate when a new request arrives for a media object for which there exists an ongoing stream [56, 4]. The server may reduce the playback rate of the earlier stream and/or increase the playback rate for the new stream (if the streams are sufficiently close). Once the two streams reach the same position in the media file, the original playback rate can be restored, and both clients can be served by a single stream. This idea can be applied hierarchically in that the resulting stream could again have its playback rate altered so that it could "catch" (or be caught by) some other stream. To ensure that clients do not notice the changes in playback rate, the playback rate variation must be limited (e.g., to ±5%). Larger performance gains can be achieved by leveraging additional available client bandwidth and storage space, as demonstrated by the *patching* [27, 25, 61, 55, 115], and *hierarchical multicast stream merging* [39, 40, 41] approaches.

2.3.3.2 Patching

Patching [27, 61, 25, 55, 115] requires that each client be capable of receiving a *unicast patch* stream from the server for the portion of the file it has missed, while listening to an ongoing multicast for the same file. The patch stream terminates when the client has received all of the missed data. Figure 2.9 illustrates how patching works when clients request the same media file at times 0, 0.1, 0.3, and 0.4, respectively, assuming a policy wherein the latter clients all listen to the full-file multicast initiated for client A as well as their unicast patch stream[3]. These patch streams are shown by solid lines in the illustration, while the "progress" of the clients (as measured by the total amount of received data) is shown by the dotted lines. Observe that the clients merge into a single group.

The various flavors of patching differ with respect to how the frequency of full-file multicast is controlled. For example, under *Greedy Patching* [27, 61], a patch stream is initiated whenever a full-file multicast is in progress. If clients are able to buffer only up to t_b time units of the media file, then for client requests arriving after t_b time units from the beginning of a full-file multicast of the requested file, only the last t_b time units of the file can be saved from multicast and later served from the buffer, resulting in long unicast patch streams and less data sharing. In contrast,

[3]In this figure, as well as Figures 2.10, 2.11, and 2.12, one unit of time on the x-axis represents the playback duration of the media file, while one unit on the y-axis represents the total amount of data in the media file.

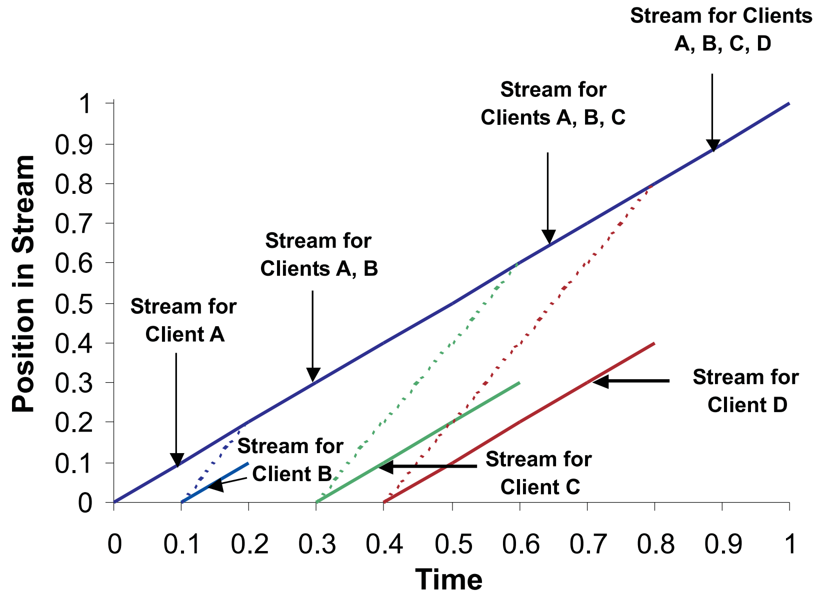


Figure 2.9: Patching

Grace Patching [61] initiates a new full-file multicast for requests that arrive t_b time units past the most recent full-file multicast. These protocols illustrate a key design tradeoff, that of the short-term gain (for the current request) of listening to the ongoing multicast, and the gains that may be experienced by future requests if a new full-file multicast is initiated. Cai *et al.* [25] and Gao *et al.* [55] independently derived the optimal threshold of full-file multicasts as a function of the client request rate and the duration of the media file. The server bandwidth requirement for an Optimal Patching policy increases with the square root of the client request rate [55, 41].

2.3.3.3 Hierarchical Multicast Stream Merging

In *hierarchical multicast stream merging* (HMSM) [41, 39, 41], clients that request the same media file are aggregated into successively larger groups, creating a binary tree merging structure. Figure 2.10 illustrates how four clients that request the same media file at arbitrary times are merged using the *closest target* policy [41]. The illustration assumes that each server stream is transmitted at the media playback rate and clients can simultaneously receive at most two such server streams. Upon

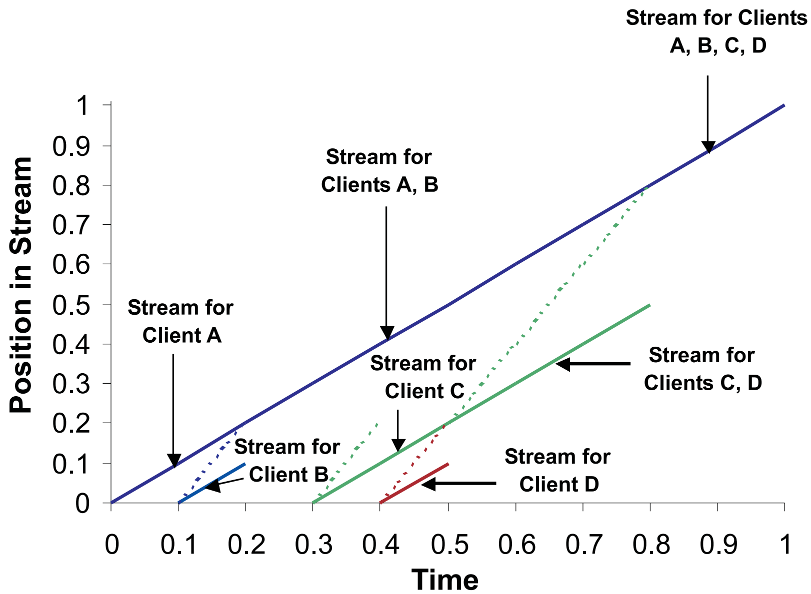


Figure 2.10: Hierarchical Multicast Stream Merging

request of the media file, each client is provided with a separate stream (depicted by a solid line) to enable immediate playback. Simultaneously, each client also listens to the closest earlier active stream. The clients' own stream is terminated when the client has received all of the data it missed from the earlier stream. The dotted lines in the figure shows the amount of data a client or a group of clients that is attempting to merge with an earlier group has accumulated (its "progress").

In the figure, client requests arrive at times 0, 0.1, 0.3, and 0.4. The streams for clients A and B merge at time 0.2, and the streams for clients C and D merge at time 0.5. Clients A and B merge with clients C and D at time 0.8. Since client arrivals are dynamic in nature, it is possible that a client's merge effort may not be successful because some other client merged with it, as shown for client C. That is, client D merged with client C before client C could merge with clients A and B. Note that the "progress point" for a group is defined according to the progress of the client with the least attained progress.

Bandwidth skimming policies proposed in [40] can apply the hierarchical merging technique illustrated in Figure 2.10 even when the achievable aggregate transmission rate to a client is less than twice the media playback rate. One such policy, called

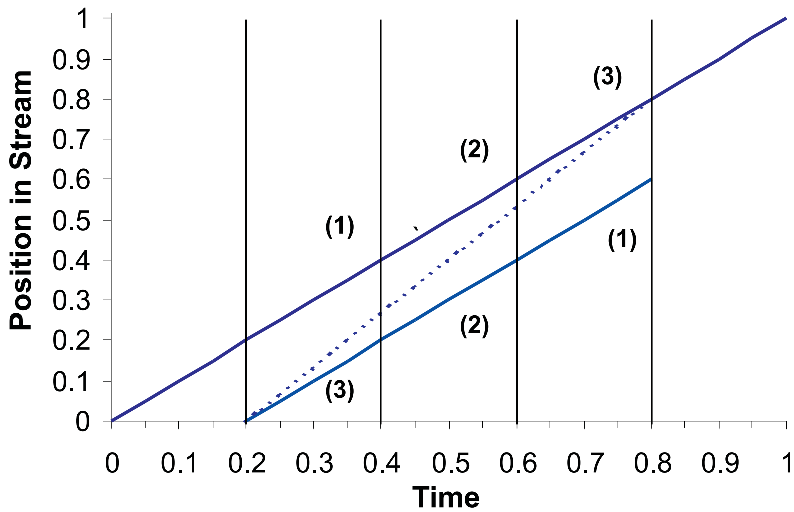


Figure 2.11: The Partition Bandwidth Skimming Policy ($k = 3$)

Partition [40], divides a stream into k channels, each at $1/k$ times the playback rate. Each channel carries $1/k$ of the media using a predefined fixed interleaving of the data packets. Assuming that a client can listen on at most $(k+1)$ channels, merging two streams requires k periods, each of duration equal to the time gap between the two corresponding requests, as illustrated in Figure 2.11 for $k = 3$. During each period, the later client listens to the number of channels from each stream indicated in parentheses near the stream. For example, in the first period, the later client listens to one of the channels of client's earlier stream, and three channels of its own stream. In the second period, the later client has already received the data that will be delivered on one of the channels transmitting the stream it initiated, so the later client listens to two channels of its own stream and two channels of the client's earlier stream.

Another policy is *Latest Patch* [40] illustrated in Figure 2.12 for three client arrivals at times 0, 0.2, and 0.3, respectively. In this policy, clients receive data from the server at their full receiving bandwidth capacity in an attempt to merge with an earlier stream. A merge occurs when a client's data stream crosses the latest playback position of the clients associated with the closest earlier stream. In the

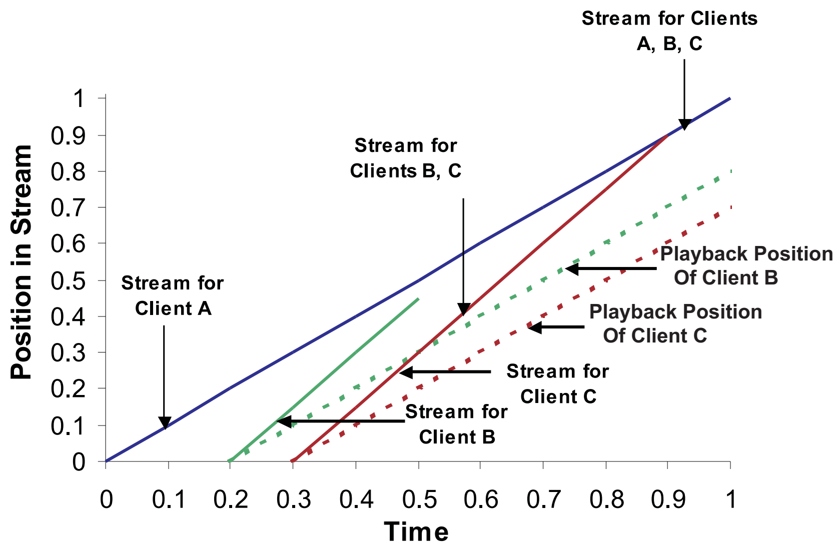


Figure 2.12: The Latest Patch Bandwidth Skimming Policy

figure, the progress of a stream as defined by the amount of data received, is shown using a solid line, while the latest playback position of the associated clients is shown using a dashed line. Note that after a merge occurs, the earlier stream is terminated. The main advantage of this policy over Partition is that it requires fewer multicast channels. A drawback, however, is the higher server bandwidth requirement since streams are shared only after a stream is completely merged.

2.3.4 Scalability Bound for Immediate Service Protocols

A tight lower bound on the required server bandwidth as a function of the client request rate for *any* protocol that provides immediate on-demand streaming of media, with no packet loss recovery, is outlined below [41]. This bound assumes that: 1) each client can receive at arbitrarily high rate; and 2) the request arrival process is Poisson as found for example in a measurement study of an educational media server [8].

Consider an arbitrary portion of a media file dx at an offset x relative to the beginning of the file. For a client request at time t, this portion of the media file can be delivered as late as time $t + x$. Therefore, all client requests that arrive between time t and time $t + x$ can share the multicast of this portion of the media file. Under the Poisson request arrival assumption, the average time from time $t+x$ until the next arrival is $1/\lambda$, implying that the minimum frequency of multicasts of this portion is $1/(x+1/\lambda)$. Therefore, the lower bound on the required server bandwidth $B^{d=0}_{min}$ (the superscript indicating that this is for *immediate service* on-demand streaming) is as follows:

$$B^{d=0}_{min} = \int_0^T \frac{dx}{x+\frac{1}{\lambda}} = \ln(T\lambda+1) = \ln(N+1). \tag{2.9}$$

Extensions of the analysis to non-Poisson arrival process is possible, yielding a very similar analytical result [41]. Note that the Poisson arrival assumption generally yields higher estimates of required server bandwidth than is the case for even "burstier" arrival processes, since higher burstiness allows greater opportunities for aggregating requests for the same file.

Figure 2.13 shows the required server bandwidth for Optimized Patching and two example bandwidth skimming protocols, together with the lower bound in Equation 2.9 on the required server bandwidth for any immediate service protocol. Note that the results for the bandwidth skimming protocol are obtained from simulations assuming Poisson request arrivals. As illustrated in this figure, the bandwidth requirement of the bandwidth skimming protocols increases only logarithmically (with a small constant factor) as the client request rate increases [40, 41]. Comparing this with Figure 2.8, note that bandwidth skimming has lower required server bandwidth than periodic broadcast protocols at low to moderate client request rates (such as under 100 requests per media playback duration). Note that even with client data rate b that is only 1.25 times the media playback rate, the server bandwidth requirement for bandwidth skimming is competitive with respect to Optimized Patching at low request rates. At high request rates, bandwidth skimming significantly outperforms Optimized Patching. Recall that Optimized Patching requires client data rate equal to twice the media playback rate. Furthermore, in contrast to both Optimized Patching and periodic broadcast, bandwidth skimming naturally allows each client to start at an arbitrary point in the media stream, and thus can support client

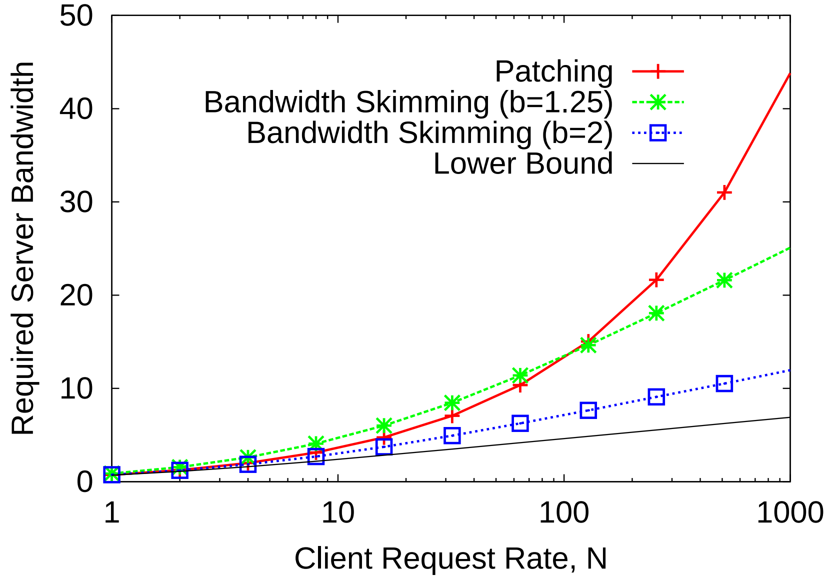


Figure 2.13: Required Server Bandwidth for Immediate Service Protocols

interactive requests including general "fast forward" requests [41].

The bound in Equation 2.9 can be extended to the case in which there is a maximum start-up delay d by adding d to the denominator of the integrated function [68]. Note that with this modification, letting $\lambda \rightarrow \infty$ yields the bound previously given for periodic broadcast schemes.

2.4 Congestion Control

Computer network architectures often follow a layered model, where each layer has well defined responsibilities with lower layers providing services to higher layers. The Internet is based on the TCP/IP protocol stack that consists of four layers: link, network, transport, and application layers [120]. The link layer is responsible for transmitting data from the host to the network via the network interface card. At the core of the TCP/IP protocol suite is the Internet Protocol (IP) that runs at the network layer, and is responsible for providing unreliable connectionless datagram delivery between hosts. The TCP/IP transport layer provides two end-to-end packet delivery protocols, namely the Transmission Control Protocol (TCP) that provides reliable connection-oriented delivery and the User Datagram Protocol (UDP) that provides unreliable connectionless delivery. Application specific protocols constitute the application layer.

A majority of applications on the Internet use TCP at the transport layer. TCP includes flow control, congestion control, and packet loss recovery mechanisms that are suitable for applications such as file transfer and the Web. Emerging multimedia applications, however, typically use UDP, because TCP's congestion control and packet loss recovery mechanisms yield highly variable transmission rates and packet delays, which are undesirable in such applications. For such applications, congestion control and packet loss recovery mechanisms must be implemented by the application or an intermediate protocol running on top of UDP.

Although this thesis is concerned with multicast applications, a number of key ideas pertaining to multicast congestion control arise from work on unicast congestion control. Section 2.4.1 presents a brief overview of TCP and related literature, followed by a discussion of TCP friendliness in Section 2.4.2. Congestion control for unicast multimedia streams is discussed in Section 2.4.3, followed by a review of rate control proposals in the multicast setting in Section 2.4.4.

2.4.1 Transmission Control Protocol

TCP provides connection-oriented, reliable, full duplex, unicast service. The sender stamps each packet with a unique sequence number and receivers return acknowledgments with the sequence number of the next expected packet, allowing the sender to identify and retransmit lost packets. TCP adapts the sending rate of the source according to perceived changes in the available network bandwidth by dynamically computing the maximum size of the *window* of unacknowledged packets in the network. The goal of TCP congestion control is to increase the window size if there appears to be additional available bandwidth, and decrease the window size when there is congestion in the network.

There are several flavors of TCP that differ with respect to packet loss recovery and congestion control, including Tahoe [63], Reno [64], Newreno [46], Sack [86], and Vegas [20, 21]. In the following sections, attention is restricted to the Reno and Vegas variants of TCP, as they are most pertinent to the work presented in this thesis.

2.4.1.1 TCP Reno

TCP Reno reacts to congestion whenever packet loss is inferred, either through a timeout, or by the receipt of three duplicate acknowledgments. The congestion control algorithm consists of two distinct phases: *slow start* and *congestion avoidance*, as outlined next.

At session start-up, the appropriate window size is not known. Jacobson proposed the slow start algorithm in which the window size is initially one (measured as number of maximally-sized packets), and in the absence of packet loss doubles every round-trip time (RTT) [63]. This exponential growth of the sending rate is achieved by increasing the window size by one whenever an acknowledgment arrives at the sender. This exponential growth continues until a packet loss is inferred, following which the congestion control algorithm either enters the congestion avoidance phase, or resumes slow start with the window size set back to one. In the congestion avoidance phase, the sender cautiously probes for additional bandwidth by slowly increasing the window size, specifically by $1/window$ for every acknowledgment received.

If packet loss is inferred through the receipt of three duplicate acknowledgments, the window is reduced by a factor of two and the sender performs *fast retransmit* by retransmitting the lost packet and enters the *fast recovery* mode. During fast recovery, the fourth duplicate acknowledgment results in the window being set to $window + 4$, and the sender continues to increase the window size by one for every subsequent duplicate acknowledgment. Receiving duplicate acknowledgments indicates that a packet has been lost from the window, yet other packets of the window have managed to reach the destination, indicating mild congestion. Thus, during fast recovery, the sender increases its window by the number of duplicate acknowledgments received. When a non-duplicate acknowledgment (also known as a "recovery" acknowledgment) is received, the window is set to the value at which the sender entered fast recovery, and normal congestion avoidance behavior resumes. TCP Reno is said to use an *additive increase, multiplicative decrease* (AIMD) algorithm because of the manner in which window adjustments are made in the absence of timeouts.

TCP Reno is known to exhibit performance problems when multiple packets are lost from a single window of packets [42]. TCP Newreno modified Reno's behavior during fast recovery on receiving a "partial" acknowledgment that acknowledges some but not all of the outstanding packets. In Reno, a partial acknowledgment takes the sender out of fast recovery, while in Newreno it retransmits the packet next in sequence to the partial acknowledgment. This modified behavior of Newreno allows recovery from multiple packet losses in the same window, by retransmitting one lost packet per RTT. Newreno remains in the fast recovery mode until all packets that were yet to be acknowledged when fast recovery was initiated have been acknowledged.

TCP uses *acknowledgment clocking* (ACK clocking), in which packet transmissions are spaced according to the spacing of returning acknowledgments. Packet losses inferred from timeouts indicates that there will likely be no further acknowledgments returning to clock future packet transmissions. Therefore, on experiencing a timeout, TCP resets its window to one and enters slow start so as to refill the pipeline in a manner that follows ACK clocking.

2.4.1.2 TCP Vegas

TCP Reno and similar variants induce losses to estimate available bandwidth and its congestion avoidance mechanism results in the window size periodically oscillating between a value that is too large and too small. TCP Vegas [20, 21] introduced a novel congestion avoidance mechanism that attempts to detect congestion in the network before packet loss occurs. Specifically, in the congestion avoidance phase, a Vegas flow estimates the number of its packets in network queues as $\theta = (\Lambda_e -$

$\Lambda_a) baseRTT$, where $\Lambda_e = W/baseRTT$ is the expected throughput of the flow, $\Lambda_a = W/R_t$ is the actual throughput of the flow, W is the current congestion window, R_t is the most recent RTT estimate, and $baseRTT$ is the minimum of all observed round-trip times. The algorithm attempts to keep the number of queued packets between α and β, the two algorithm parameters (also referred to as threshold parameters) by increasing the window size by one every RTT if $\theta < \alpha$, and decreasing the window size by one if $\theta > \beta$. The Vegas congestion avoidance mechanism potentially provides smoother transmission rates and less packet loss. Other innovations of Vegas are concerned with the slow start and retransmission mechanism, and are not discussed here. The interested reader is referred to [20, 5, 21] for details.

Simulation studies have investigated the performance of TCP Vegas under a wide variety of network conditions (e.g., see [90, 112]). It has been shown that the throughput achieved by a Reno flow exhibits a much stronger RTT bias than a Vegas flow. It is also known that the relative aggressiveness of Vegas flows depends on the threshold parameters and the buffer space available at the bottleneck queue. Choosing threshold parameters that allow Vegas flows to compete fairly with other instances of Vegas flows as well as flows employing more aggressive TCP variants is a hard problem, and has been an impediment towards large scale deployment of Vegas.

2.4.2 TCP Friendliness for Multimedia Streams

It is highly desirable that all types of Internet traffic share resources fairly, both between flows of the same type and between flows of different types. One definition of fairness states that the long-term bandwidth usage of a non-TCP flow should not exceed that of a conformant TCP flow under the same network conditions [44]. Several protocols have been proposed for unicast multimedia streaming that succeed in achieving this goal to varying degrees, and are discussed in Section 2.4.3.

For multicast flows, there is no unanimously accepted definition of fairness. Some researchers argue that a single multicast session deserves no greater throughput than a competing TCP flow under similar network conditions, while others argue that the bandwidth share of the multicast session should be a function of the number of receivers in the session [129, 131, 82]. The rate control component of this thesis considers the former more conservative definition of fairness.

Designing TCP-friendly rate control for multicast multimedia flows is complicated for several reasons. First, for both unicast and multicast multimedia flows, there is a tradeoff between the goal of reacting promptly to changes in congestion, and the goal of minimizing the variability in the quality of media obtained by receivers. Second, unlike a unicast multimedia flow that can adjust its transmission rate based on the feedback received from the client, feedback from multiple clients can result in excessive load at the sender. Third, the heterogeneity of the client transmission paths implies that there exists no universally accepted rate at which the sender should transmit.

An approach to handling some of the above mentioned issues is *layered* multicast, where a media file is divided into a base layer and a number of enhancement layers [117]. Each receiver performs congestion control through its choice of how many layers to receive. An alternative technique is *simulcast* that involves multicasting a number of different quality versions of the same media file. Each receiver decides which version to receive based on its achievable reception rate [29]. Note that TCP flows behind a common bottleneck link, but with different round-trip times, typically achieve different throughputs. For a multicast session, it is desirable that receivers behind a common bottleneck link get the same number of layers regardless of their round-trip times, thus complicating the definition of TCP friendliness. Also, with either layered multicast or simulcast, the possible changes in transmission rate are limited according to the rates of the available layers/versions.

2.4.3 Rate Control for Unicast Multimedia Streams

One approach to congestion control in the unicast context involves use of AIMD algorithms [118, 105, 13]. The *Rate Adaptation Protocol* (RAP) proposed by Rejaie *et al.* is an example of an AIMD-based scheme in which the source performs rate adaptation based on acknowledgments sent by the receivers [105]. The acknowledgments are used to detect packet losses and estimate RTT. Rate adaptation is performed once per RTT, with the transmission rate being increased linearly in the absence of loss, and decreased multiplicatively when congestion is detected. RAP uses the ratio of short-term to long-term averages of RTT to fine tune the sending rate. Further, the authors advocate using RAP with layered media and using work-ahead and receiver buffering to absorb short-term fluctuations in the sending rate without adding/dropping layers [104].

Loss-Delay Based Adjustment (LDA) is another AIMD-based scheme proposed by Sisalem and Schulzrinne [118]. Bansal and Balakrishnan have proposed *binomial* congestion control algorithms that require the window size to increase inversely proportional to a power of k of the current window size and decrease the window proportional to a power of l of the current window size, such that $k + l = 1$ [13]. Binomial congestion control algorithms are shown to be fair to TCP and can be used to control the characteristic rate fluctuations of AIMD schemes like RAP and LDA.

An alternative approach to the AIMD schemes is to use a TCP throughput model to determine the transmission rate of the sender [94, 123, 45]. The most mature example of such *equation-based* schemes is the *TCP-Friendly Rate Control* (TFRC) protocol [45]. In TFRC, the receiver periodically returns the time-average loss event rate to the sender. On receiving feedback, the sender can also estimate RTT, and then use a TCP throughput model to determine the fair share of bandwidth. It is argued that use of equation-based congestion control with suitable parameterization of the TCP throughput function allows the transmission rate to be gradually changed, while still being responsive to congestion in the network.

2.4.4 Multicast Rate Control

A protocol for congestion control in the context of layered media multicast was first introduced by McCanne *et al.* [88]. In the *Receiver-driven Layered Multicast* (RLM) protocol, each video layer is transmitted on a separate multicast group. Receivers drop a layer when congestion is detected. Periodically, receivers probe for additional bandwidth by carrying out *join experiments*. A join experiment is successful if no losses are encountered after adding an additional layer. Outcomes of join experiments are communicated to other receivers using a scalable technique called *shared learning*.

The RLM protocol has been found to have several drawbacks [73]. First, uncoordinated join experiments can result in unfairness and instability. Second, multicast group leave requests incur substantial latencies (on the order of a few seconds) with the current Internet protocols [126], and therefore failed join experiments can cause the network to remain in a state of congestion for several seconds. Another cause of concern is unfairness with other RLM flows and with TCP flows. These and several other issues have been addressed by subsequent research efforts.

The *Receiver-driven Layered Congestion* (RLC) protocol proposed by Vicisano *et al.* [126] addresses some of the above mentioned drawbacks of RLM. To synchronize join experiments, the sender places *synchronization points* in the data stream, which dictate when a receiver can join additional layers. A synchronization point on layer i permits a join experiment for all layers $j \leq i$. This mechanism attempts to keep receivers behind a common bottleneck link synchronized. Another key feature of the RLC protocol is the use of periodic bursts of traffic to test for additional capacity. The burst on layer i results in a data rate for subscribing to layers 0 through $i + 1$. Receivers interpret packet loss during such a burst as a signal that subscription to layer $i + 1$ would not be suitable. The layering scheme assumed by RLC is such that dropping a single layer results in a multiplicative bandwidth reduction. RLC tries to emulate TCP behavior at longer time scales by controlling the frequency of synchronization points across the layers. Since requests to leave a multicast group can incur a substantial latency, RLC introduced the notion of a *dead period* after initiating a layer drop, during which receivers do not react to any congestion signals.

The bandwidth discovery in RLC relies on periodic bursts of traffic being successful in overflowing router queues, whenever insufficient bandwidth exists to support the additional layer. However, choosing the right burst duration may be difficult in practice [73, 22].

Fair Layer Increase/Decrease with Dynamic Layering (FLID-DL) [22] generalized the RLC protocol and proposed a clever technique for dealing with large multicast leave latencies. FLID-DL uses a *dynamic layering* scheme, where each multicast group cycles among the layers, as well as an off period, in the order highest to lowest layer, followed by the off period, followed by the highest layer again. The off period duration is large enough to absorb the worst case leave latency. To maintain a steady subscription level, each receiver must join the multicast group that will be transmitting its highest subscribed layer by the time the layer assignment cycles. Layers can be dropped by not performing any joins. FLID-DL, like its predecessors, assumes a cumulative data layering scheme (i.e., receivers subscribe to all layers $j \leq i$, for some i). The data rate of a receiver subscribed to layers 0 through i is assumed to be given by $R_i = c^i R_0$, where R_0 is the rate of the base layer, and the constant $c > 1$. Like RLC, FLID-DL uses synchronization points that are cumulative. Receivers can perform join experiments when the appropriate synchronization point is received. A layer is dropped by the receiver upon encountering packet loss. The frequency of synchronization points is computed so that a mathematical model of the protocol behavior yields the throughput as a function of loss rate similar to that predicted by a TCP Reno throughput model for some fixed RTT.

The use of a fixed RTT has the disadvantage of being possibly either aggressive or not as aggressive as a TCP flow under similar conditions, depending on the actual RTT witnessed by a TCP flow between the same end points. Another drawback is that the protocol does not react to changes in queuing delays, as does TFRC for example.

An approach in which each receiver explicitly uses a TCP throughput equation to determine its layer subscription was first proposed by Turletti *et al.* in the context of audio streaming [125]. A similar protocol was proposed by Tan and Zakhor for video streaming [122]. A more elaborate protocol was recently proposed by Sisalem and Wolisz, in which the server dynamically tunes the rates of the layers using feedback received from the clients [119]. These schemes use the RTP Control Protocol (RTCP) [113] to obtain estimates of RTT for use in a TCP throughput equation. A disadvantage of using actual RTT values, however, is that receivers behind a common bottleneck with differing round-trip propagation delays may subscribe to differing numbers of layers, which is undesirable.

The *Wave and Equation Based Rate Control* (WEBRC) protocol is an equation-based multirate congestion control scheme [82]. Two key characteristics of this protocol include transmitting data in *waves* wherein the transmission rate on a multicast quickly increases and then decreases exponentially over time, and using a multicast analogue of the unicast RTT called Multicast Round-Trip Time (MRTT) in place of RTT in the TCP throughput equation.

The MRTT is defined as the time between a multicast group join and the reception of the first packet from the group. For a single receiver session, the MRTT equals the actual RTT, neglecting join processing delays (which may be significant). With multiple receivers participating in a multicast group, receivers joining earlier will measure higher MRTT values than those joining later, as join requests need only travel part way to the server till they reach a branch of the multicast tree. However, owing to the cyclic wave structure of data transmission, receivers that join a group early in one cycle measure a higher MRTT and compute a lower desired reception rate, and will join later in the next cycle. The sum of the MRTT measurements of all the receivers is claimed to approximately equal the sum of the link delays of the multicast delivery tree. Note, however, that individual MRTT values are proportional to the number of receivers in the session, and thus the throughput share of the multicast session increases with the number of receivers.

It should be noted that the wave-like data transmission scheme of WEBRC allows fine-grained bandwidth reception rate changes by joining a particular multicast group earlier or later than it did in the last cycle. The WEBRC protocol is insensitive to large leave latencies and the frequency of joins/leaves is less compared to FLID-DL [82].

An example of an ideal protocol is the Packet pair receiver-driven Layered Multicast (PLM) [74]. In this protocol, sender sends packets in pairs, and receivers use the spacing between the two packets to estimate the bandwidth between the sender and the receiver. This protocol's bandwidth inference process does not induce losses, with receivers fairly sharing bandwidth with TCP as well as quickly converging to their respective fair share bandwidths. Although this mechanism for bandwidth estimation is simple, it does make the assumption that the network implements Fair Queuing.

More recently, single rate multicast congestion control schemes have been proposed that are either AIMD-based or equation-based [107, 109, 132]. A key challenge in designing such rate control schemes is to develop a scalable mechanism for sending feedback from a large group of clients to the source such that the sending rate can be determined. Several recently proposed multicast congestion control schemes have been surveyed in [75, 131].

2.5 Error Control

A simple error control strategy is for the receivers to request retransmissions of lost packets by the sender. Such retransmission based schemes may not be suitable for applications such as on-demand streaming for several reasons. First, these schemes have to ensure that there is sufficient time for the client to request a lost packet, and the retransmitted data to arrive at the client prior to its playout point through conservative choice of the playout delay. Note that it is hard to estimate the retransmission time because of the variable queuing delays in the Internet. Second, many receivers may concurrently request retransmissions, resulting in feedback implosion at the sender. Third, it is generally agreed that server retransmission based schemes do not scale in terms of the required server bandwidth for applications such as on-demand streaming media [84]. One possible solution to these difficulties is to utilize distributed "repair" servers [102]. In this section, however, attention is restricted to error recovery mechanisms that do not utilize retransmissions. For a discussion on retransmission based recovery schemes, the reader is referred to [26, 100, 80].

2.5.1 Error Concealment

Error concealment techniques are used by receivers to mask the effect of missing packets without any assistance from the sender [100]. These schemes were originally developed for voice data transmission, and in that context rely on the assumption that human speech characteristics do not significantly change over short time periods. For example, a simple scheme is to repeat playout of the last correctly received packet. Other possibilities include silence substitution, or noise substitution, in place of the lost packet. A replacement for the lost packet can be generated by applying more complex interpolation and pattern matching techniques on the packets surrounding the loss. These techniques achieve better speech reconstruction, but are computationally more expensive.

In environments with relatively low loss rates, receiver-based local error concealment techniques can be used to mask the effect of missing packets quite effectively [100]. However, in environments with significant rates of burst losses in which multiple packets may be lost, error concealment techniques become inadequate for many applications [26].

2.5.2 Forward Error Correction

In Forward Error Correction (FEC) schemes, the sender transmits redundant data that allow the receiver to reconstruct missing packets. At the cost of additional bandwidth, either "approximate" or "exact" replacements of the lost packets can be obtained. FEC has been previously applied to live or scheduled broadcasts, and unicast streaming applications, but not to one-to-many on-demand streaming applications.

A simple FEC scheme is to transmit redundant data with each transmitted packet (e.g., a redundant copy of packet $(n-1)$ can be bundled with packet n). In the context of multimedia streaming, the data can be media data that is encoded at a lower rate [100]. Packet loss results in use of redundant data and in lower quality playback. Another approach is to generate a parity packet across each group of n packets [100]. The sender transmits each parity packet following the corresponding data packet, and receivers can independently recover from any single packet loss. Interleaving can be used to reduce the impact of burst losses.

Another approach is to use *erasure codes* [108, 24, 100] that construct multiple independent redundant packets for each set of data packets. A (n, k) erasure code takes k source packets and generates n $(n > k)$ encoded packets. To simplify the decoding of the source data and allow partial recovery in high loss situations, the set of encoded packets can include the source packets. Unless more than $(n - k)$ packets are dropped, the receiver can reconstruct the original data stream. Erasure codes are more scalable than retransmission based schemes, since different receivers can recover from different lost packets using the same redundant data.

The encoding/decoding complexity of erasure codes can be significant. One factor to consider is the computational complexity of the encoding/decoding schemes. Another factor to consider is the "decode efficiency" of the erasure codes, where decode efficiency is defined as the ratio of the number of packets needed to decode the original data with respect to the number of packets in the original data. The Reed-Solomon codes [108] have optimal decode efficiency of 1, but have high decoding times. Recently proposed Tornado codes [24] have suboptimal decode efficiencies (e.g., on average equal to 1.05), but provide manifold improvement in decoding times. Another consideration is the overhead of transmitting redundant data.

Chapter 3

Packet Loss Recovery

One of the principal contributions of this thesis is the development of *scalable* on-demand streaming protocols that can deliver media files to a group of heterogeneous clients in a *reliable* fashion. These protocols are applicable for any networking infrastructure that involves clients with substantially different bandwidth capabilities and packet loss characteristics, such as the Internet and wireless delivery platforms. Although this problem has been effectively solved in the context of bulk data delivery (e.g., one to many file transfers as required for large scale software updates) [110, 126, 24], the solutions proposed there cannot be directly applied to scalable on-demand streaming protocols, as it results in unacceptable start-up latencies.

The *digital fountain* [24] approach [110, 126, 24] is designed to deliver bulk data reliably in diverse environments, such as the Internet, satellite networks, and ad hoc networks. In this approach, a server constructs an infinite stream of distinct packets from the original file using an *erasure code*[1]. These distinct packets are multicast whenever there are any clients listening to the session, allowing receivers to reconstruct the original file from any subset of the distinct packets equal to the total number of packets in the source data. This approach has several desirable properties, such as flexibility, efficiency, and scalability. Note that clients may start

[1]In practice, it is not possible to generate an infinite set of distinct packets given k source packets. The usual approach is to generate n distinct packets from k source packets and repeatedly cycle through these encoded packets. Byers *et al.* [24] suggest setting $n = 2k$ to approximate a digital fountain.

downloading the file at any desired time. Further, clients can stop listening to the multicast channel and resume at a later time to obtain the remaining number of required packets. This method is efficient because both the amount of data and the processing time required to reconstruct the original data is minimal. Scalability is inherent in this approach as no feedback channels are required to recover lost data, and the required server bandwidth is a single server channel that is independent of the client population.

When applied to on-demand streaming, the digital fountain approach results in high start-up latency because clients are unable to reconstruct the portion of the media file required to begin playback until the data for the entire media file is obtained. Scalable on-demand streaming protocols discussed in Chapter 2 do not address the issue of reliable delivery over lossy networks. As described later, these protocols are not amenable to the digital fountain approach because of the timing constraints inherent in their media transmission schedules. In low loss rate environments, local error concealment strategies, such as repeating the last correctly received frame, may be adequate. New protocols are required to handle higher loss rates, since error concealment techniques are rendered ineffective [100].

This chapter develops new periodic broadcast and bandwidth skimming protocols for scalable, reliable, on-demand streaming in environments with heterogeneous client access characteristics and packet loss rates. The new protocols, namely Reliable Periodic Broadcast (RPB) and Reliable Bandwidth Skimming (RBS) represent a synthesis of the scalability of the on-demand streaming protocols and the reliability of the digital fountain approach.

While previous periodic broadcast protocols require that the aggregate transmission rate to each client be equal to twice the media playback rate, the new RPB protocols can support clients with maximum aggregate transmission rate equal to a small percentage (e.g., 25%) greater than the media playback rate. The new RPB protocols support arbitrarily high request rates for the media using server bandwidth that grows only logarithmically with the ratio of media playback duration to the start-up latency, requires no feedback from the client to the server, and can efficiently handle clients with heterogeneous packet loss rates.

The RBS protocols require minimal client-server interaction, work for client data rates less than twice the media playback rate, and have almost immediate playback. These protocols require server bandwidth that grows logarithmically with the client request rate, and are suitable for media files that have time varying popularity. Further, the RBS protocol can easily support interactive client requests, such as rewind, fast forward, pause, and resume. The RBS protocol, however, is not as effective as RPB in handling bursty losses.

The remainder of this chapter is organized as follows. Section 3.1 discusses the difficulties in incorporating reliability in previously proposed scalable on-demand streaming protocols and outlines the key requirements of the new protocols. Section 3.2 presents a qualitative and quantitative analysis of alternative packet loss recovery techniques. The RPB and RBS protocols are developed and analyzed in Section 3.3 and 3.4, respectively. A prototype implementation of the RBS protocol is discussed in Section 3.5. Section 3.6 summarizes the chapter.

3.1 Motivation

This section will discuss why the application of the digital fountain approach to previously proposed scalable on-demand streaming protocols is not straightforward.

The periodic broadcast protocols proposed in [127, 2, 54, 62] cannot apply the digital fountain approach since segments are played back as they are received. The Harmonic Broadcasting protocol with deterministic start-up delay described in Section 2.3.1 (see page 29) and the protocols in [69, 60, 96, 59] can transmit each segment as a digital fountain. However, the aggregate transmission rate to the clients for these protocols equals the server bandwidth requirements (i.e., 5 to 10 times the media playback rate). The focus of this work is on protocols that are suitable in a multicast scenario where clients listen to only a small subset of the sever channels and yet achieve scalability close to the lower bound. Another advantage of this approach is that a larger fraction of the available transmission bandwidth to the client

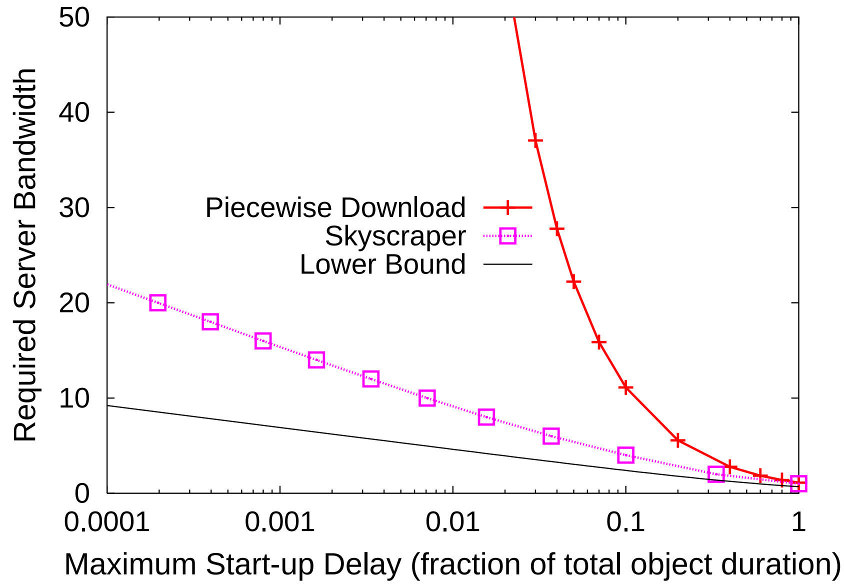


Figure 3.1: Comparing Alternative Approaches to On-demand Streaming

can be used to deliver a higher quality media object. For example, if a client can receive data at 500 Kbps, then the Skyscraper system could only stream a media file that has playback rate 250 Kbps, whereas a protocol that requires client data rate of only 1.25 times the playback rate can stream a media file that has playback rate equal to 400Kbps.

Reliable on-demand streaming can be achieved by a simple periodic broadcast protocol, referred here as "Piecewise Download", which divides a media file into equal size segments. Each segment is delivered as a digital fountain at a rate equal to $1/(1-p)$ times the media playback rate, where p is the assumed client loss rate. A client downloads one segment at a time, playing the previously downloaded segment while receiving the next segment. The start-up delay equals the playback duration of a segment. This approach requires server bandwidth that grows linearly with a linear decrease in start-up delay. The performance gap between existing scalable streaming protocols that do not provide any reliability and the simple "Piecewise Download" approach is illustrated in Figure 3.1, assuming $p = 10\%$ and perfect decode efficiency[2]. For comparison purposes, the figure also shows the lower bound

[2]Decode efficiency is defined as the ratio of the number of packets needed to decode the original

on any protocol with a given start-up delay.

Recall that both patching (Section 2.3.3.2, page 33) and bandwidth skimming (Section 2.3.3.3, page 34) initiate a new stream in response to a new client request for a media file, and therefore each stream in progress is required for immediate playback by some client. It is not clear how these protocols could be modified to incorporate packet loss recovery in a scalable fashion.

The properties desired in the new streaming protocols are:

- **On-demand**: Clients should be able to begin playback of the requested media after a small adjustable start-up delay.
- **Tolerant**: The protocols should support a client population with wide ranging heterogeneity with respect to packet loss rates and end-to-end throughput.
- **Reliable**: The protocols should allow reconstruction of each media packet before it is needed for playback for clients with packet loss rates less than or equal to the loss rate the system is designed to support.
- **Flexible**: The protocols should allow the clients the ability to obtain a media quality of their choosing by allowing additional start-up delays. Alternatively, the protocols should allow the clients to receive a lower quality media file in the presence of higher packet losses.
- **Efficient**: The protocols should require minimal client feedback, and the total amount of data each client receives should be minimal.
- **Scalable**: The server bandwidth requirement of the protocols should be close to the minimum possible server bandwidth for any protocol that satisfies the above goals.

---

data with respect to the number of packets in the original data.

3.2 Loss Recovery Strategies

In environments with relatively low loss rates, simple receiver-based local error concealment techniques [100] can be used to mask the affect of missing packets. However, in environments that have bursty losses [98], such as the Internet, multiple consecutive packets may be lost, and error concealment techniques become inadequate for many applications [26].

This section considers three possible server-based packet loss recovery techniques for on-demand media streaming, namely unicast retransmissions, multicast retransmissions, and erasure codes. These techniques are qualitatively compared in Section 3.2.1. Section 3.2.2 presents bounds for these alternative packet loss strategies, followed by a quantitative comparison of these techniques in Section 3.2.3.

3.2.1 Qualitative Discussion

An important consideration for adopting any recovery strategy is the implementation complexity. For erasure codes, it is necessary to consider the efficiency of the encoding/decoding operations. Retransmissions, on the other hand, have to factor the unpredictable end-to-end transmission delays in the implementations, as well as consider techniques for handling feedback implosion [50, 79, 26, 49, 77, 134, 75].

Closely related to the implementation complexity is the start-up delay experienced by a client. Start-up delay is defined as the time between initiation of the client request and the time at which the client can finally begin playback. Clearly, this start-up delay is the sum of any latency associated with the protocol (e.g., latencies in periodic broadcast schemes), end-to-end latencies and jitter, and overheads associated with implementations. Erasure codes will incorporate decoding times in the start-up delay, while retransmissions will incorporate retransmission delays. Although, it is conceivable that the start-up delays might be comparable for these alternative techniques, it is easy to believe that the start-up delay for an erasure code approach is more predictable.

Another important consideration is the scalability of these alternative techniques measured in terms of the server bandwidth requirements. The required server bandwidth for immediate service protocols can be measured as a function of packet loss rates, and client request rates. For bounded delay protocols, the required server bandwidth is measured as a function of start-up delay and packet loss rate. Note that in the multicast streaming context, different clients may experience different packet losses, and the effectiveness of the recovery strategy depends on sharing of the redundant data. Obviously, unicast retransmissions incur the highest bandwidth cost since there is no sharing of the retransmitted data. The advantage, however, is that each client receives only the data it needs. Efficiency of retransmissions can be improved by allowing multicast retransmissions of redundant data. Thus, multiple clients can recover a particular lost packet using a single retransmission. The disadvantage is that many clients may receive more data than they need to recover from their own losses. Multicast transmission of redundant data generated by application of erasure codes enables a single redundant packet to repair different losses for different clients, thus intuitively implying that this approach may be more scalable than the retransmission-based approaches.

3.2.2 Scalability Bounds

The scalability of the alternative approaches for reliable on-demand streaming are compared by developing lower bounds on the required server bandwidth for the recovery strategies considered, assuming that the packet loss rate is p. These bounds are based on the bounds reviewed in Section 2.3.4 (page 37) that assume no packet loss [54, 16, 41]. The notations used here are shown in Table 3.1.

Consider the multicast of erasure-coded data. In this case, a client needs to receive encoded data equal in amount to the size of the file. Assuming that packet loss experienced by each client is on average equal to p, the server must transmit, on average $1/(1-p)$ times more data than when packet loss recovery is not implemented. Thus, the lower bound $B_{min}^{d=0,e}$ for immediate service protocols using erasure codes

Table 3.1: Notation for the New Scalability Bounds

Symbol	**Definition**
λ	average client request rate for a media object
T	media object playback duration
N	average number of requests for the object that arrive during a period of length T ($N = \lambda T$)
d	maximum start-up delay for object playback
B	required server bandwidth (in units of the object playback bit rate)
p	packet loss probability

(indicated by the second superscript) is given by:

$$B_{min}^{d=0,e} = \frac{1}{1-p} \ln(N+1). \tag{3.1}$$

Similarly, the lower bound for bounded-delay protocols using erasure codes $B_{min}^{d>0,e}$ is given by

$$B_{min}^{d>0,e} = \frac{1}{1-p} \ln(\frac{T}{d}+1). \tag{3.2}$$

The bound for unicast retransmissions is derived next. For an object of duration T minutes, the average amount of retransmissions per client is $\frac{pT}{1-p}$, measured in units of playback duration of the media file. Thus, for an average request rate of λ, the required server bandwidth for retransmissions is $\lambda\frac{pT}{1-p}$. The total required server bandwidth, therefore, equals the sum of the bandwidth needed for retransmissions and the bandwidth for transmitting the data packets (see equation 2.9). Substituting $N = \lambda T$, the lower bound $B_{min}^{d=0,ur}$ for immediate service protocols using unicast retransmissions (indicated by the the second superscript in the notation) is given by

$$B_{min}^{d=0,ur} = \ln(N+1) + \frac{p}{1-p}N. \tag{3.3}$$

A lower bound on the required server bandwidth for multicast retransmissions is derived in [84] under the assumptions of Poisson request arrivals and independent packet loss probability p. The derivation is quite complex, and the bound is not

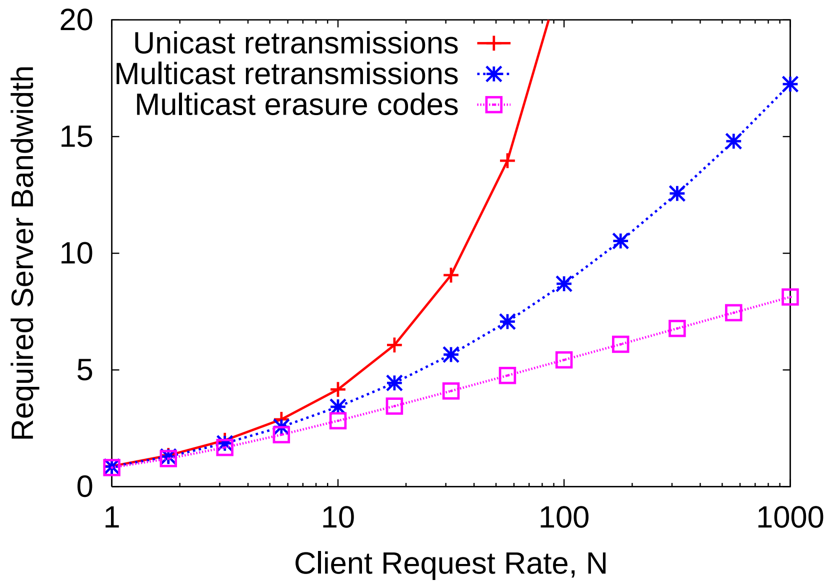


Figure 3.2: B_{min} for Alternative Packet Loss Recovery Techniques ($p = 0.15$)

obtained in closed form. The lower bound can be obtained by solving the expression for particular values of N and p.

3.2.3 Numerical Results

Numerical results for the immediate service scalability bounds as a function of normalized client request rate N, assuming 15% average packet loss rate, are presented in Figure 3.2. As expected, the required server bandwidth for unicast retransmissions increases exponentially with an increase in the client request rate. The results also show that a multicast retransmission-based recovery strategy has significantly higher server bandwidth requirement than schemes using erasure codes for moderate to high client request rates (e.g., $N > 50$). Qualitatively similar differences in the bounds are observed for lower values of p. These observations have motivated the use of erasure coding in the protocols developed in this chapter. The following sections will demonstrate that the lower bounds for packet loss recovery using erasure codes can actually be attained by practical on-demand streaming protocols.

Table 3.2: Parameters of the New Reliable Periodic Broadcast Protocols

Symbol	Definition
K	total number of segments
r	segment transmission rate (in units of the object playback bit rate)
s	maximum number of streams that clients listen to concurrently ($s \leq K$)
b	maximum aggregate transmission rate to a client (in units of the object play rate), $b = s \times r > 1$
l_k	playback duration of the k^{th} segment (relative to the length of segment 1)

3.3 Reliable Periodic Broadcast

This section develops optimized periodic broadcast protocols that satisfy the goals outlined in Section 3.1. Table 3.2 defines the notations used for the protocols developed in this section. The main assumptions of the protocols are: 1) the maximum aggregate transmission rate to the clients b, is a parameter that can be set to a small multiple of the media playback rate; 2) playback for a segment begins only after the segment is entirely downloaded; and 3) clients can successfully recover from packet losses up to a tunable loss rate p. Design of these protocols requires addressing several fundamental challenges, thus necessitating a four-stage development process.

The Optimized Periodic Broadcast (Optimized PB) protocols developed in Section 3.3.1 do not support packet loss recovery, but provide fundamental insight regarding how maximally growing segment sizes can be designed for periodic broadcast protocols where clients concurrently listen to s segment transmissions each streaming at rate r, under the assumption that each segment is completely received before playback begins. Recall that the Skyscraper Broadcasting protocol plays segments as they are received, thus requiring segment transmission rates to at least equal the media playback rate. A key advantage of entirely downloading a segment before playback is that the segment transmission rate r can be arbitrarily lower than the playback rate. This provides the opportunity of exploring the performance of periodic broadcasts for situations in which the aggregate transmission rates to clients are less than twice the media playback rate.

Section 3.3.2 develops a family of basic Reliable Periodic Broadcast (Basic RPB) protocols that extend the Optimized PB protocols to support packet loss recovery. Each segment k is erasure-coded and transmitted as a digital fountain. Clients can reconstruct each segment k prior to the segment playback deadline, provided that no segment experiences loss rates greater than a tunable parameter p. The performance evaluation of the protocol shows that the required server bandwidth can closely match that predicted by the lower bound for reliable delivery with erasure codes.

Section 3.3.3 addresses the impact of bursty packet loss common in environments such as the Internet [98], by allowing each segment to have a different cumulative loss protection. This section also discusses how lower loss on an earlier segment can enable sustaining higher loss for a later segment.

Finally, Section 3.3.4 discusses how clients can tradeoff delay and available bandwidth to the client to sustain loss rates higher than the chosen value of p.

3.3.1 Optimized PB Protocols

The new Optimized PB protocols assume that:

1. packet losses are rare and can be effectively masked by local error concealment techniques;

2. clients can simultaneously receive a maximum of s multicast streams, each being transmitted at a fixed rate r; and

3. each segment is fully downloaded before the segment playback begins.

Figure 3.3 illustrates the working of the Optimized PB protocol when the media file is partitioned into 6 segments, with each segment being repeatedly multicast on its own channel at the media playback rate. Immediately after tuning in, clients start listening on the first s channels, as illustrated in the figure by the shaded regions for the client arrival marked by the arrow. After completely receiving segment 1,

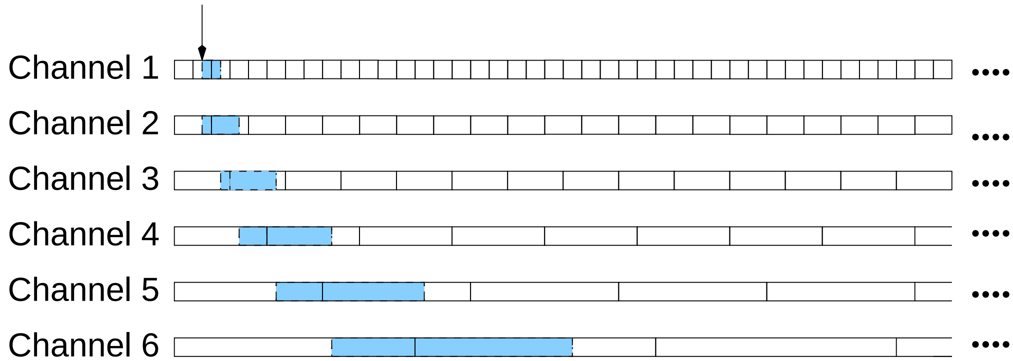


Figure 3.3: Example Optimized PB Protocol ($K = 6$, $r = 1$, $s = 2$)

the client tunes in to receive segment 3, and after completely receiving segment 3 starts receiving segment 5, and so on. Since data constituting a segment is often received out of order, the minimal possible start-up delay for this protocol equals the time required to download the first segment, as the segments can be played back only after it is completely downloaded. For this start-up delay, the segment size progression guarantees the receipt of each other segment in its entirety just in time to guarantee playback without pauses.

For periodic broadcast protocols, faster growth in segment size progression translates into lower start-up delays. Assuming that each segment is transmitted at the media playback rate (i.e., $r = 1$), the maximum possible segment size progression can be derived as follows. Given that a client can concurrently listen to a maximum of s channels, the length of each segment k, $1 < k \leq s$, must equal the time to download segment 1 plus the time to playback segments 1 through $k-1$. Clients start receiving segment $k > s$, as soon as the download of segment $k-s$ is complete. This download must complete as soon as segment $k-1$ finishes playback. Assuming the length of the first segment, $l_1 = 1$, relative segment lengths can be given as:

$$l_k = \begin{cases} l_1 + \sum_{j=1}^{k-1} l_j & 1 < k \leq s \\ \sum_{j=k-s}^{k-1} l_j & k > s. \end{cases} \tag{3.4}$$

The above segment size progression can be generalized for segment transmission rate $r < 1$ by noting that in this case, the time to download a segment increases by a factor of $1/r$. Therefore, the modified segment size progression can be given as:

$$\frac{1}{r}l_k = \begin{cases} \frac{1}{r}l_1 + \sum_{j=1}^{k-1} l_j & 1 < k \leq s \\ \sum_{j=k-s}^{k-1} l_j & k > s. \end{cases} \tag{3.5}$$

The main advantage of segment transmission rates less than the media playback rate is that the aggregate transmission rate to each client, $b = s \times r > 1$, can be less than two times the media playback rate. As described earlier, this allows a higher fraction of the client bandwidth to be devoted to obtaining higher quality media.

An Optimized PB system designed with a fixed number of segments K uses server bandwidth $B = r \times K$. The deterministic start-up delay experienced by clients is $\frac{T}{r \sum l_k}$.

Figure 3.4 compares the performance of the Optimized PB protocol and the Skyscraper Broadcasting [62] protocol. For both protocols, the required server bandwidth is plotted as a function of start-up delay, assuming $b = 2$. Both the average and the maximum start-up delays are shown for the Skyscraper systems since the start-up delay for this system is non-deterministic. For the Skyscraper protocol, the maximum start-up delay is twice the average start-up delay. The following key conclusions can be drawn from the results. First, the performance of the Optimized PB system is competitive with Skyscraper Broadcasting, although Optimized PB requires that a segment be fully received before playback can begin. Second, the performance of the Optimized PB protocol improves with decreasing segment transmission rates. This second observation is explored in detail next.

Figure 3.5 compares the frequency of multicast of portions of the media file for Optimized PB systems that approximately have the same start-up delay, but have differing segment transmission rate, r. The frequency of multicast is defined as the number of times a particular portion of the media file has to be transmitted per play-

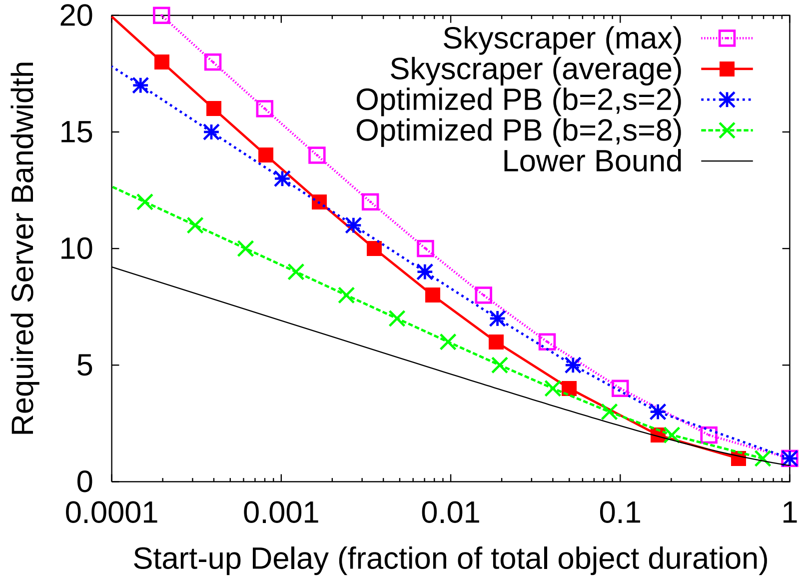


Figure 3.4: Performance Comparison of Optimized PB and Skyscraper Broadcasting

back duration. The lower bound on the multicast frequency can be derived from the observations made for deriving the lower bound on the required server bandwidth. That is, a portion of the media file at an arbitrary position x is transmitted at frequency $1/(x+d)$, where d is the start-up delay. Note that the lower bound derivation assumes infinitesimally small segments, and hence the plot for the lower bound is a smooth curve. However, the plots for the Optimized PB are step functions, reflecting the fact that an entire segment is received ahead of playback and thus the end of the segment is received much before it is actually needed for playback. As r decreases, the plots for the Optimized PB become smoother because the number of segments increases, correspondingly decreasing the segment sizes. Although, the $r = 0.02$ line closely tracks the lower bound, a small gap can be noticed between the Optimized system and the lower bound. This gap is due to the finite bandwidth assumption ($b = 2$) of the protocol, while the lower bound assumes infinite client receive bandwidth.

The maximum client buffer space requirement of the protocol is derived next. As with other other periodic broadcast protocols, note that the client buffer requirement keeps growing until the rate of draining exceeds the rate of filling. This condition

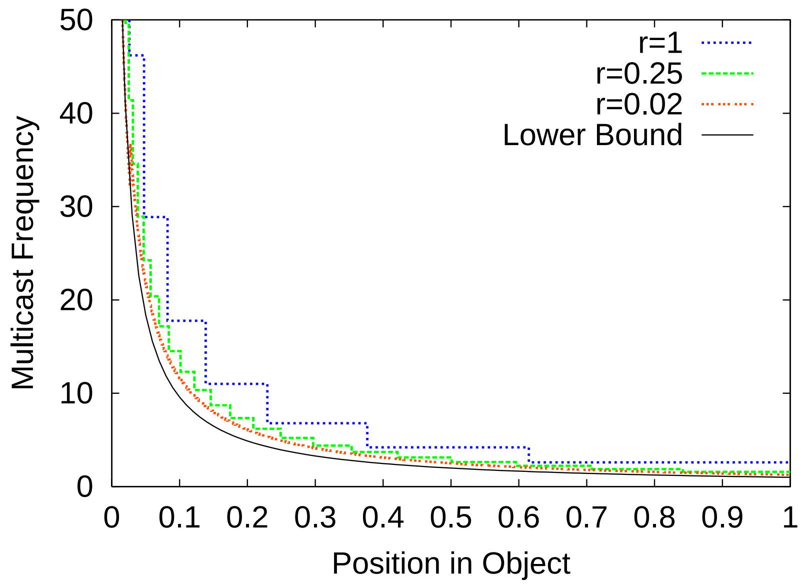


Figure 3.5: Multicast Frequency vs. Object Position ($b = 2$, $d \approx 0.0043T$)

is satisfied when the download of segment $j^* = \max\left(1, K - \lfloor \frac{1}{r} \rfloor\right)$, is completed. At this point, the download of segment j^* is complete, while segments $j^* + 1$ through K are partially downloaded. Noting that the partial download for any segment $j^*+1 \leq i \leq K$ contains all but the amount of data that would have been downloaded during the playbacks of segments j^* through $i-1$, the following expression of buffer space requirement can be obtained, expressed as a fraction of the media file size:

$$\frac{l_{j^*} + \sum_{i=j^*+1}^{K} \left(l_i - r \sum_{k=j^*}^{i-1} l_k \right)}{\sum_{k=1}^{k=K} l_k}. \tag{3.6}$$

Figure 3.6 shows the required buffer space for the Optimized PB protocol for different combinations of b and r. For the purpose of comparison, the client buffer space requirement for the Skyscraper protocol is also shown. The buffer space requirement for the Skyscraper protocol is bounded by the size of the last segment [62]. It is interesting to note that despite of entirely obtaining a segment before playback, the buffer space requirement for the Optimized PB protocol is competitive with that of Skyscraper, especially for lower values of start-up delays and segment transmission rates $r < 1$.

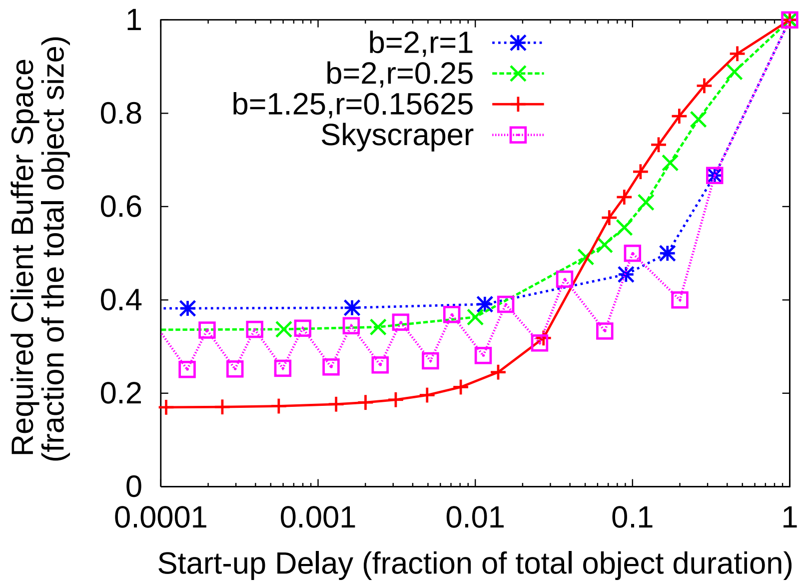


Figure 3.6: Buffer Space Requirements for Optimized PB and Skyscraper Protocols

3.3.2 Basic Reliable Periodic Broadcast

This section generalizes the Optimized PB protocols to accommodate perfect recovery of a segment assuming that the "cumulative packet loss" at the end of receiving a segment is at most p. Since multiple segments are received in parallel, cumulative packet loss is defined as the average packet loss over the current segment k, and all other segments that must be completely received before reception of the segment under consideration can begin. Since reception of any segment k begins after segment $k-s$ is completely received, cumulative loss for segment k is defined as the average packet loss over the reception of segments $k, k-s, k-2s, \cdots$. Later sections will describe how the protocol can be extended to support environments that have packet loss greater than the assumed upper bound p.

Since each segment is completely received before playback begins, the Optimized PB protocol can be extended to support packet loss recovery by delivering each segment as a digital fountain. With this modification, erasure codes are applied on each segment to generate an (infinite) stream of encoded packets. Encoded packets corresponding to a particular segment are transmitted on individual channels, instead of cyclically broadcasting the same segment, and clients simply listen to a channel

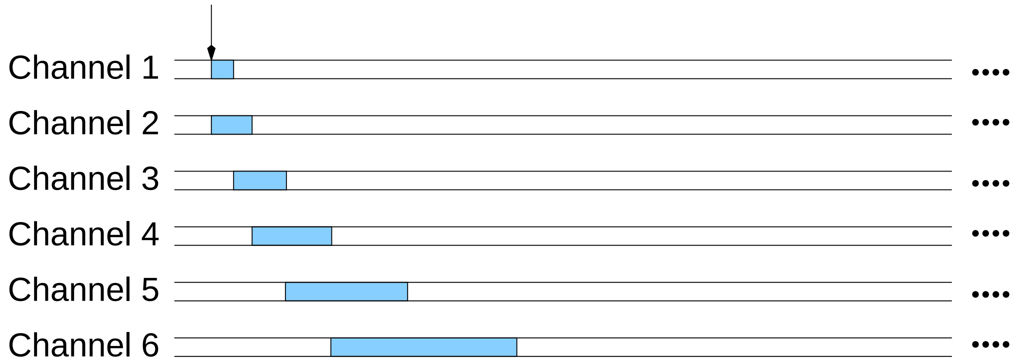


Figure 3.7: Example Optimized RPB Protocol ($K = 6$, $r = 1$, $s = 2$, $p = 0.1$)

until it has received the packets required to reconstruct the segment. Assuming that clients experience packet loss with probability p while receiving a segment, the download time for that segment is a factor $a = 1/(1-p)$ times longer than when there are no packet losses. Assuming that the maximum packet loss is p, and segment decoding times are negligible, the maximum relative segment sizes can be given as:

$$a \times \frac{l_k}{r} = \begin{cases} a \times \frac{l_1}{r} + \sum_{j=1}^{k-1} l_j & 1 < k \leq s \\ \sum_{j=k-s}^{k-1} l_j & k > s. \end{cases} \tag{3.7}$$

For a given number of server channels K, the deterministic start-up delay is $\frac{aT}{r \sum l_k}$.

Figure 3.7 illustrates the reception sequence for a client arrival marked by the arrow for the Optimized RPB protocol with parameters $K = 6, r = 1, s = 2$, and $p = 0.1$, under the assumption that each segment exactly experiences packet loss rate of p. A key difference with respect to the Optimized PB protocol in Figure 3.3 is that the notion of segment boundaries do *not* exist in the Optimized RPB protocol since each channel now transmits a stream of erasure-coded packets that are of equal value to the client (i.e., if the original segment was composed of n packets, the client can reconstruct the segment from any n erasure-coded packets). Note that the download time for a particular segment depends on the loss rate experienced while the segment is being downloaded. Another important property of the protocol is that less packet loss while downloading an earlier segment can provide greater tolerance to packet loss while receiving a later segment, provided the cumulative packet loss does not exceed the tunable parameter p.

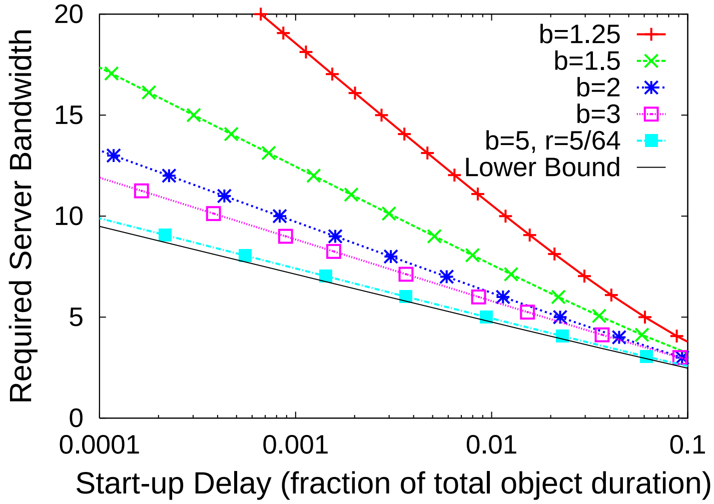
Figure 3.8 shows the impact of p and b on the required server bandwidth as a function of the start-up delay, for delivering a single media file. At low and moderate loss rates (e.g., $\leq 10\%$), it is possible to achieve a start-up delay equal to 1% of the total playback duration, even for clients with aggregate transmission rate only 1.25 times the media playrate. The figures also show that for fixed s, the required server bandwidth of the Optimized RPB protocol approaches the lower bound on the required server bandwidth given by equation (3.2) as the aggregate transmission rate to the client increases.

Figure 3.9 illustrates the impact of decreasing r for a fixed b, on the required server bandwidth. It is clear that decreasing r improves performance. However, the relative performance improvement as r becomes less than 0.25 is small. Also note that decreasing r for a given server bandwidth results in an increased number of smaller sized segments. While smaller segments may be favorable because of lower decoding times, it comes at the cost of managing more multicast channels.

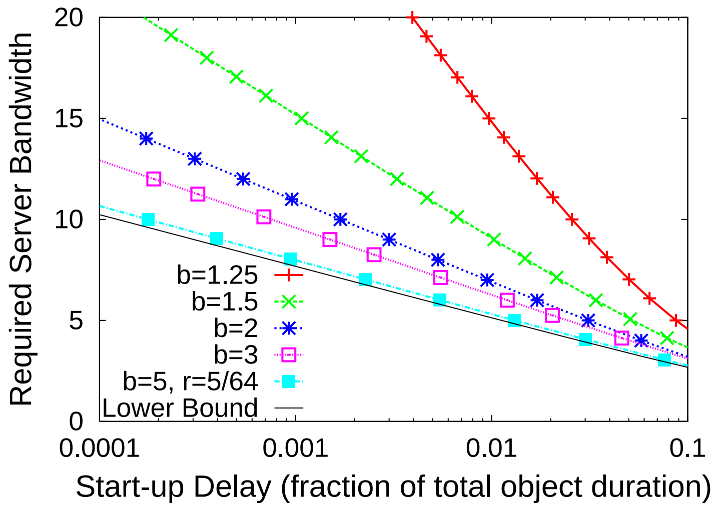
3.3.3 Accommodating Bursty Packet Loss

In the RPB protocol, clients that suffer cumulative packet loss less than or equal to p at the end of a segment download are able to provide jitter-free playback. Further, the protocol allows clients to perform "work-ahead" by receiving segments before they are needed when the cumulative loss rate is less than p, thus accommodating a subsequent segment with a higher packet loss than p. This works as long as the cumulative loss at the end of the segment download is less than or equal to p. The fact that clients can begin segment reception at any arbitrary point in time is the key feature that allows for this tradeoff.

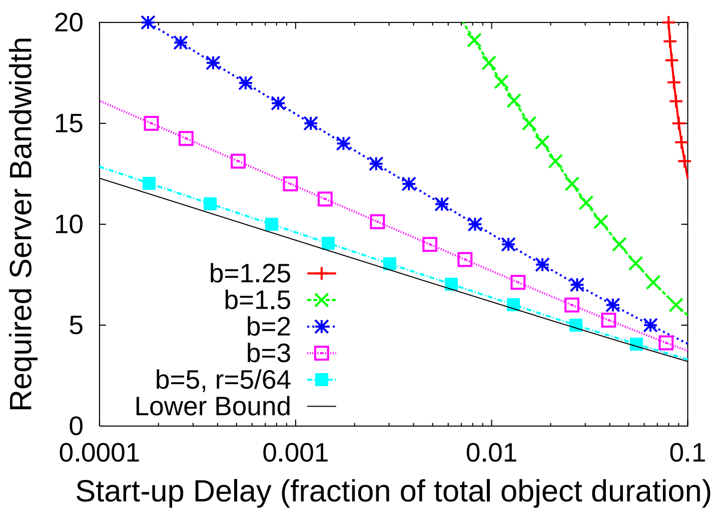
Previous Internet measurement studies indicate that packet loss is typically bursty [98, 26, 18, 67]. Sudden loss spikes will most likely affect the earlier seg-



(a) 3% Packet Loss



(b) 10% Packet Loss



(c) 25% Packet Loss

Figure 3.8: Performance of the Basic RPB Protocols ($a = \frac{1}{1-p}$, default $r = b/8$)

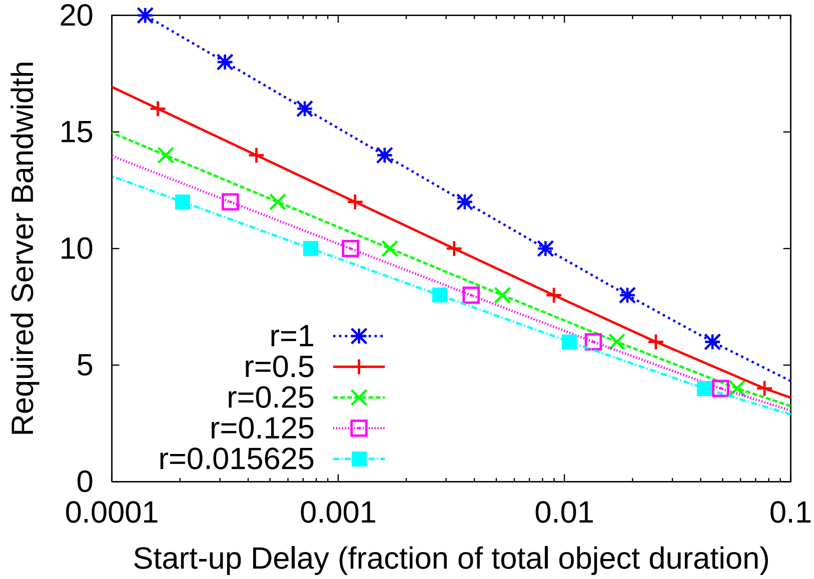
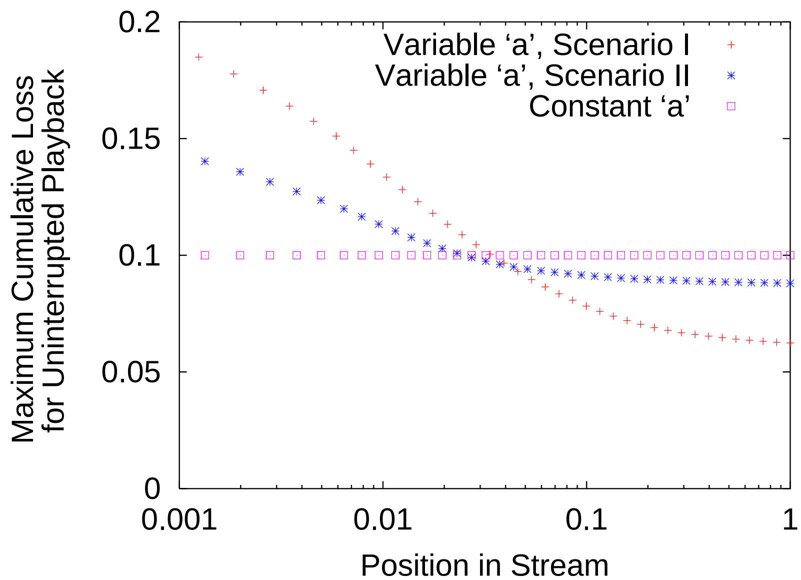


Figure 3.9: Impact of Streaming Rate on Basic RPB Performance ($b = 2$, $p = 0.1$)

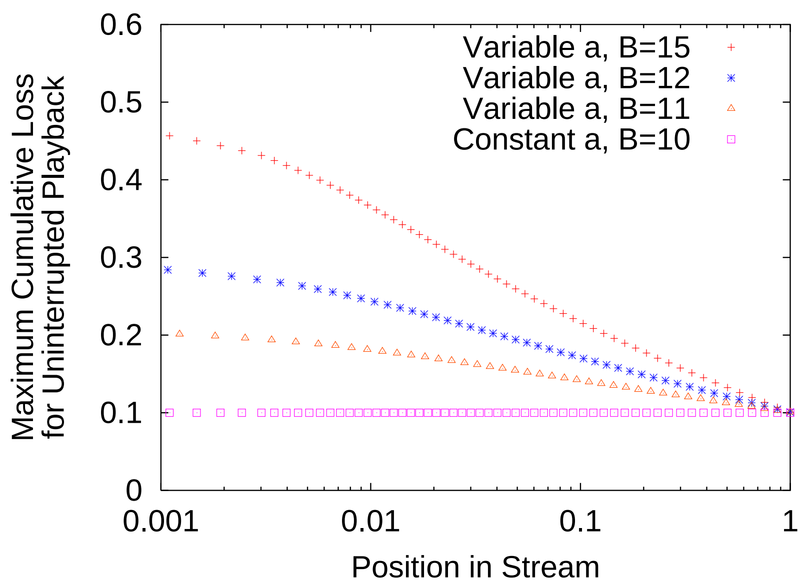
ments that are smaller in size than the later segments, and since "work-ahead" is most effective for the later segments, it is desirable that the earlier segments have a higher level of protection against packet loss. This can be accomplished by allowing a longer download time for a segment k by using a different a_k value. The segment size progression with this modification is easily derived as:

$$a_k \times \frac{l_k}{r} = \begin{cases} a_1 \times \frac{l_1}{r} + \sum_{j=1}^{k-1} l_j & 1 < k \leq s \\ \sum_{j=k-s}^{k-1} l_j & k > s. \end{cases} \tag{3.8}$$

Figure 3.10 illustrates two strategies for providing greater protection to earlier segments against packet loss, with Basic RPB systems that have fixed server bandwidth $B = 10$, segment transmission rate $r = 0.25$, start-up delay $d = 0.00017T$, and require clients to receive data at rate $b = 2$. For example, Figure 3.10(a) considers two scenarios that provide higher cumulative loss tolerance for the initial segments by trading off loss tolerance for the later segments. Figure 3.10(b), on the other hand, illustrates scenarios that have a fixed overall protection with higher protection for initial segments, provided at the cost of additional server bandwidth.



(a) Fixed Server Bandwidth ($B = 10$)



(b) Fixed Overall Loss Protection

Figure 3.10: Loss Tolerance using Variable a ($b = 2, r = 0.25, d = 0.00017T$)

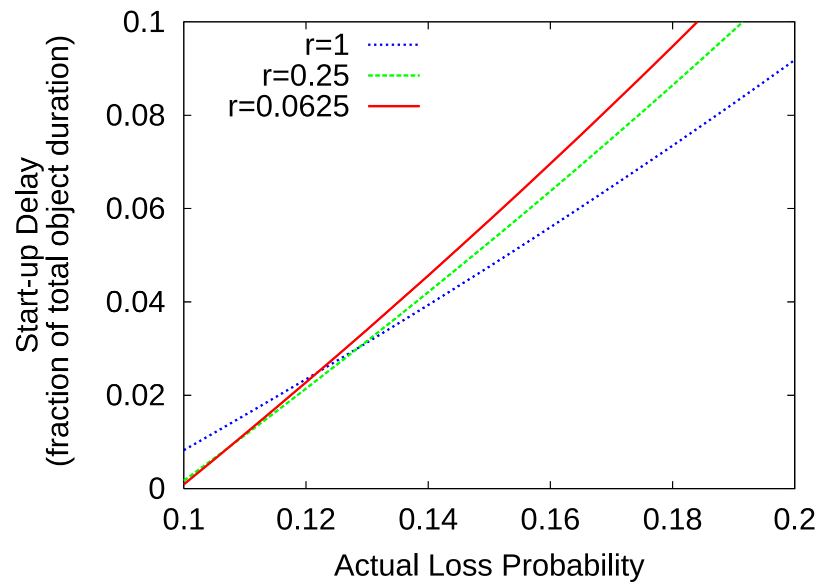
3.3.4 Client Heterogeneity

The RPB protocols developed up to this point assume that the client population is homogeneous with respect to loss rates on the transmission paths from the server to the clients and the achievable client data rates. This section demonstrates how the protocols can accommodate a limited degree of client heterogeneity. The mechanisms described here can be used in conjunction with other complementary mechanisms for accommodating client heterogeneity. For example, redundant network transmission paths [78] can be used to keep the cumulative loss rate seen by the clients below the selected parameter value p. In addition, the server can use layered multicast [75][3] to enable clients with higher/lower aggregate transmission rates to receive correspondingly higher/lower quality media. In the simplest case, each layer of the media file could be delivered using the same protocol parameters, with clients dynamically determining which layers to receive using a rate control [88, 126, 75, 22] policy.

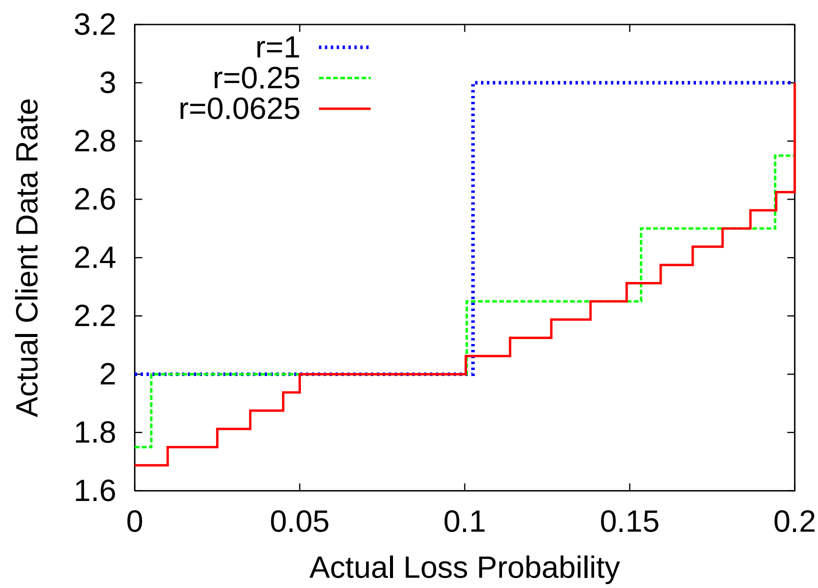
Figure 3.11 provides examples of limited tradeoffs between client start-up delays, packet loss rates, and client data rates. Figure 3.11 (a) illustrates how a client that experiences a loss rate higher than the RPB protocol specified sustainable loss rate can achieve uninterrupted playback by increasing its start-up delay. Note that the client will somehow have to obtain a good estimate of the loss rate to employ this strategy. A simple strategy might be to use the loss rate experienced by the client while downloading the first segment (i.e., the first segment and portions of the next $s-1$ segments that are being downloaded in parallel) as an estimate, and delay starting playback if the loss rate is above acceptable limits.

As another example, Figure 3.11(b) illustrates how clients with higher data rates can support higher loss rates by listening to more streams. In this example, the default RPB system parameters are $B = 10, b = 2, p = 0.1$. The curve shows plots for three different streaming rates (i.e., different r values). The graph shows the required client data rates as a step function of the actual loss rate since client

[3]Equivalently, the server could use the Simulcast approach [29, 75], where the server stores different "versions" of the media file, each being of a different quality.



(a) Start-up Delay vs. Loss Probability



(b) Client Data Rate vs. Loss Probability

Figure 3.11: RPB Performance for Heterogeneous Clients ($B = 10, b = 2, p = 0.1$)

bandwidth increments at fixed intervals of r. In general, a client data rate greater than twice the media playback rate is required to support higher loss rates. The results also show that clients that experience lower than the anticipated loss rate can listen to fewer than the specified s channels. Further, it appears that clients can accommodate higher loss rates for the same client data rate if the channel streaming rate is smaller.

3.4 Reliable Bandwidth Skimming

The Reliable Bandwidth Skimming (RBS) protocol developed in this section has certain advantages over the RPB protocols. First, the bandwidth usage of the RBS protocol decreases with reduction in the client request rates, thus making them more feasible for environments where the request rates vary significantly. Second, the RBS protocols are more suitable for supporting client interactivity. Third, the RBS protocols provide (almost) immediate playback upon initiation of the request by the client. However, there are some disadvantages to RBS as well. For example, the RPB protocol may have less tolerance to bursty losses. Further, handling client heterogeneity with respect to loss rates and client data rates appears to be extremely difficult.

The RBS protocol partitions a media file into several small segments and uses erasure codes to "stretch" the segments (i.e., generate encoded packets greater in number than those in the original segment) that can recover from a fixed amount of loss per segment. That is, if a segment consists of n packets and the maximum allowed loss rate per segment is p, then a total of $n/(1-p)$ encoded packets are generated. These encoded packets are transmitted on two separate streams, with n encoded packets being transmitted at the media playback rate on the *primary* stream, and the remaining $np/(1-p)$ packets being transmitted at rate $p/(1-p)$ times the media playback rate on the *secondary* stream. The starting time of the secondary stream is offset with respect to the primary stream to allow some resilience to bursty packet loss. This is necessitated by the small segment lengths, which in turn are required to preserve the immediate service feature of the bandwidth skimming protocols [41]. Note that the offset determines the latest time by which encoded packets corresponding to a segment may be obtained to enable segment reconstruction.

In the RBS protocol, both the primary and the secondary streams can be merged with other primary and secondary streams, respectively, using the same stream merging principle as in the previously proposed bandwidth skimming protocols [41]. As an example, consider implementing RBS using the Partition [40] merging policy. Both the primary and secondary streams are divided into a number of substreams k such that clients can receive $k+1$ primary and $k+1$ secondary substreams (i.e., $b = (1+1/k)(1+p/(1-p))$). The main innovation in the RBS protocols is the separation between primary and secondary streams and the effective sharing of the secondary streams among many clients.

Figure 3.12 illustrates the performance of the RBS protocol for $p = 0.1$ and various values of client data rate b. These results are obtained from simulations under Poisson request arrivals. Qualitatively similar results were obtained for a heavy-tailed distribution of interrequest times modeled by a Pareto distribution. All results in the figure have 95% confidence intervals that are within 5% of the reported values. For comparison purposes, the graphs also show the lower bound from equation (3.1), which assumes infinite client data rate. The results show that the RBS protocols adapt to variations in the client request rate, and yield performance reasonably close to the lower bound.

Comparing Figures 3.12 and 3.8(b), it can be seen that the required server bandwidth for the RBS protocol with request rate N is within a small constant factor of the required server bandwidth for an RPB protocol with similar assumed client data rate and a start-up delay equal to $\frac{1}{\lambda}$, as might be predicted from the bounds on the required server bandwidth (Section 2.3.4, page 37). For example, the required server bandwidth at $N = 100$ for the RBS protocol with assumed client data rate $b = 2.22$ and maximum loss rate of 10% is 7.7, while the required server bandwidth for the RPB protocol with assumed client data rate $b = 2$, $r = 0.25$, $p = 0.1$, and

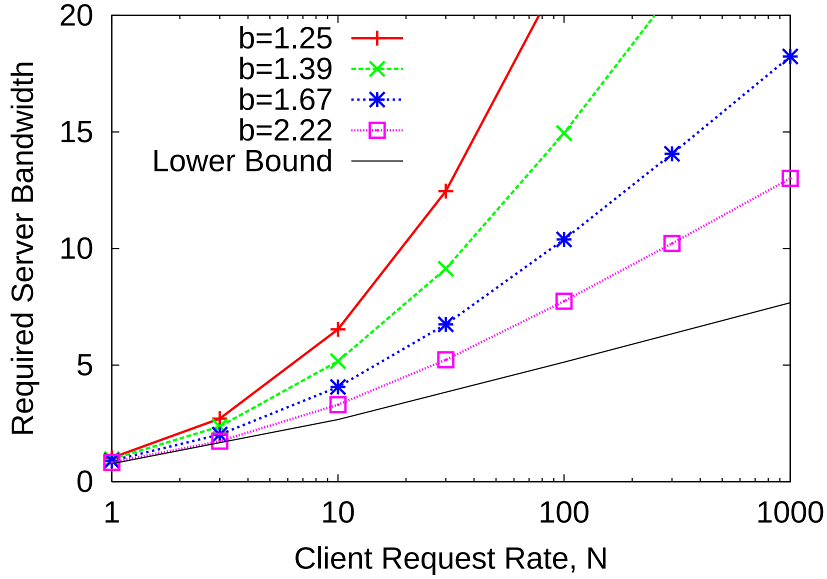


Figure 3.12: RBS Performance ($p = 0.1$)

start-up delay of 0.01 (as a fraction of the total object duration), is 7.0.

3.5 Prototype Implementation

This section discusses an implementation of the RBS protocol, presented here as a "proof of concept" of the scalable streaming protocols developed in this chapter. The implementation extends the SWORD[4] prototype, which was developed by researchers at the Universities of Wisconsin, Saskatchewan, and Washington to experiment with scalable bandwidth skimming protocols [121]. The SWORD prototype has been in production use with the eTeach server[5] at the University of Wisconsin, Madison, streaming videos for online courses.

The prototype supports delivery of Windows media files and is implemented using "client-side" and "server-side" proxies. The scalable streaming protocols are implemented between the proxies and are transparent to the actual Windows media server and clients. This design has the advantage of requiring no modifications to commercial media players and servers. To facilitate a better understanding of the

[4]SWORD is an acronym for Scalable Wide-area On-demand Reliable Digital Streaming.
[5]For more information, please see `http://eteach.engr.wisc.edu/`.

prototype design, a discussion of how the Windows media player interacts with a server using the HTTP protocol is first presented in Section 3.5.1. Section 3.5.2 will describe the design and working of the SWORD prototype. The RBS implementation is described in Section 3.5.3, followed by a summary of experiments in Section 3.5.4.

3.5.1 Accessing a Media File using HTTP

The SWORD prototype exploits the fact that the Windows media server can be instructed to use the HTTP protocol for media file delivery. The following request/response sequence has been reported in [121] for streaming a Windows media file using the HTTP protocol:

1. The user clicks on a hyperlink to a media metafile (i.e., a .asx file), using a Web browser. The browser sends an HTTP GET request for the requested file.

2. The Web server responds with an HTTP OK message, and encapsulates the contents of the .asx file in the body of the message. The message body contains the URL of the requested media content file (i.e., a .asf file). The HTTP response also includes a `content_type` header that indicates the specific media application to be initialized by the client browser.

3. The browser examines the `content_type` header and launches the appropriate media player (i.e., the Windows media player in this case).

4. The media player sends a GET request for the URL of the media content file.

5. The media server replies with an OK message. The Windows media server also includes some binary information in the body of the HTTP response that presumably contains some information for the media player.

6. The player sends another HTTP GET request for the same media file, but with some other information in the pragma field.

7. The server responds with an HTTP OK message, with the contents of the .asf file sent in the body.

Note that no further communication from the client to the server is necessary if the player receives all data on time. Otherwise, the player sends an HTTP POST message to the server indicating the problem. The server can then take corrective action (e.g., reduce the rate at which data is sent).

3.5.2 The SWORD Prototype

The SWORD prototype is implemented as a pair of proxies (Open source Apache Proxy Server Version 1.3.), one typically located at the "client-side" of the network and the other located at the "server-side". This design, illustrated in Figure 3.13, has the desirable property of allowing experimentation with new delivery protocols without requiring modifications to proprietary client/server systems. Currently, as deployed with eTeach, the users simply download the client component of the software and configure their browsers to use this client-side proxy. This client-side proxy is configured to interact with the server-side proxy (which, in eTeach, is placed on the same machine as the Media server).

The HTTP request/response messages exchanged between a media player and the server are transparently forwarded by the proxy modules of the SWORD software until a request for media content is received (i.e., the second GET request for a .asf file). At this point, the server component passes control to the WWSTP[6] server module. The WWSTP server module sends the request for the content to the media server. The resulting HTTP OK response from the media server is appended with a "use WWSTP" field and sent to the SWORD client component. Upon receiving the HTTP OK response with the "use WWSTP" field, the SWORD client drops into the WWSTP client module. The client component also sends the HTTP OK response back to the player. Note that at this point, the media server is sending

[6]The transport protocol used by the SWORD prototype is called "Wisconsin Washington Saskatchewan Transport Protocol" to reflect the names of the universities collaborating in this project.

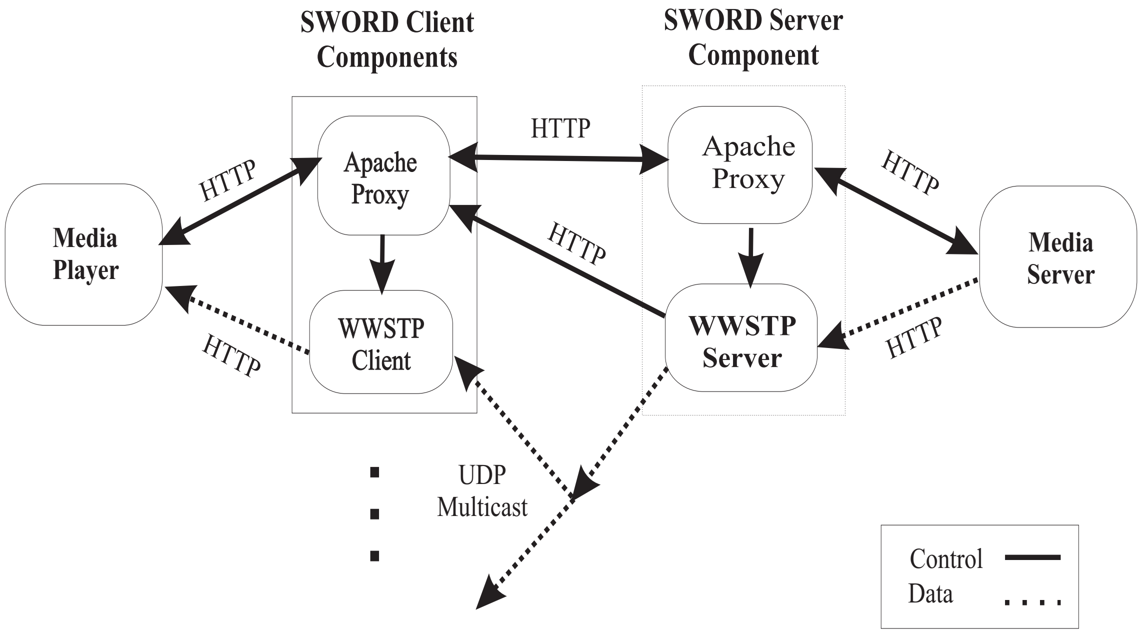


Figure 3.13: The SWORD Prototype Architecture

the media file to the SWORD server component. This data will be buffered at the server component and will be sent using multicast as defined by the scalable streaming protocol in use (currently, an HMSM protocol). Details of this process are described next, and illustrated in Figure 3.14.

An IP multicast group is set up by the server component after sending the HTTP OK response. Then, a `New Client` message is sent to the client component that provides the client with the address and the port number of this group. The client component joins this multicast group and informs the server that it is ready to receive data by sending a `Listening` message. Since overheads associated with setting up multicast routing trees can delay proper delivery of data packets from the server to the client, the server performs a simple multicast test by pinging the client component. The client component responds with a `Joined` message on receiving a ping packet from the server on the multicast group. After receiving this Joined message, the server starts forwarding data on the multicast group.

Note that the SWORD server might already be receiving the media file from the actual server, while trying to establish the multicast connection between to the SWORD client. This data is buffered at the server, and stored according to

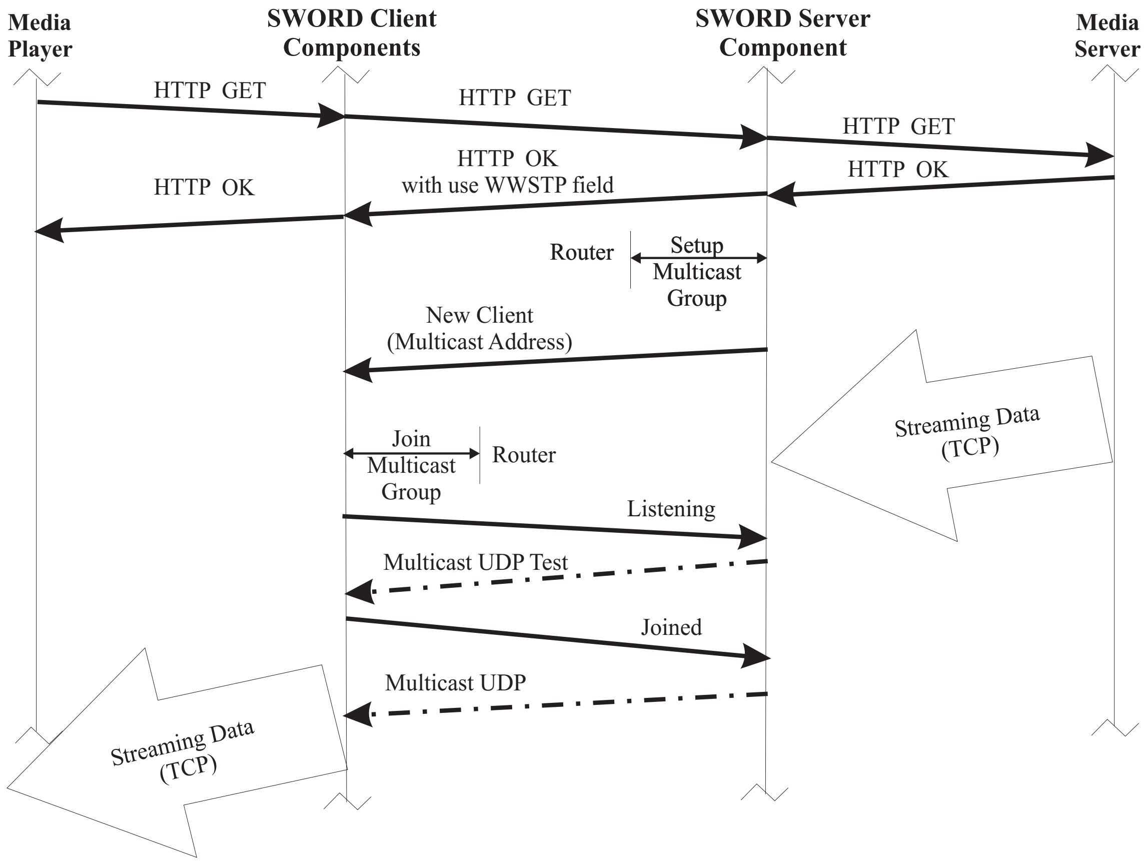


Figure 3.14: SWORD Client/Server Interaction

the timestamps applied by the media server[7]. Once the connection to the client is established, data is sent out by the server at regular intervals (e.g., once every 50 milliseconds in the current implementation). A limit is placed on the amount of data that can be sent in one interval to reduce the chances of overflowing network and client buffers.

Once the SWORD client starts receiving data, it builds a small buffer to handle network delay jitter (usually 1 to 3 seconds worth of media data), and then starts to forward the data to the media player. Experimentation with an earlier version of the prototype in Fall 2001 revealed that the media player is very sensitive to packet losses within a group of packets with the same timestamp. If any packet is lost in the group, referred to as the timestamp group, the whole group must be dropped,

[7]The actual timestamps applied by the Windows media server to the binary data stream are retrieved by analyzing the data stream.

thus highlighting the need for packet loss recovery techniques.

The server component runs the HMSM algorithm at regular intervals to determine how various streams for the same media file may be merged. Specifically, merging a later stream with an earlier initiated stream requires sending an `Update` message to the clients listening to the later stream with the multicast address and port number of the earlier stream. These clients start listening to the earlier stream, and send a `First Group` message back to the server that provides the timestamp of the first data packet received on this newly joined stream. This information is used by the server to determine when the merge is completed, and the later stream can be terminated.

3.5.3 The RBS Implementation

A RBS implementation is available in the prototype, wherein the data stream received by the WWSTP server module is encoded before being streamed, and decoded before sending the data to the media player. For the purpose of encoding/decoding data packets, a publicly available Reed-Solomon based erasure coding software is used [108]. An important design decision concerns determining the appropriate segment size for encoding, and the factor by which these segments should be stretched.

As mentioned earlier, the media player is very sensitive to packet loss within a timestamp group. The number of data packets in a timestamp group can vary, and therefore, generating segments with a fixed number of packets (for encoding/decoding purposes) is difficult and problematic. The approach used here considers variable-sized segments, where segments are composed of packets that constitute an integral number of timestamp groups. Specifically, the segment size for encoding is defined in *real time*, and is an integral multiple of the interval used for sending data. For example, if a segment consists of four data intervals of 50 milliseconds each, the server will encode 200 milliseconds of binary data at a time. The encoding module guarantees that packets with the same timestamp are not split into different segments.

The factor by which a segment is stretched is determined as follows. Suppose that a segment consists of N UDP packets. Then, the transmission burst size without any additional parity packets is $\lceil N/I \rceil$, where I is the number of data intervals in the segment. The stretch factor for the encoding is defined in units of burst size. Thus, to recover from the loss of L bursts, K parity packets are required such that $K \geq L\lceil (N+K)/I \rceil$. Note that the additional parity packets increase the transmission burst size to $\lceil (N+K)/I \rceil$

The merging algorithm remains largely unaltered. That is, individual streams can be merged based on the timestamp of the first data packet received on the stream that is being "caught".

The client component buffers encoded packets and periodically checks the buffer to determine whether or not it can decode a segment. Note that no decoding overhead is incurred if all of the data packets in a segment are received. The client component also ascertains when a segment cannot be decoded, and discards the packets belonging to this segment from its buffer. Furthermore, in the event that a segment can not be decoded, the client module attempts to retrieve any available complete timestamp groups from the undecodeable segment.

When streams are trying to merge, the WWSTP client uses data from its own stream and not from the stream it is trying to merge with, until the merge is complete. This is important as attempted merges may not be successful for a number of reasons (e.g., the stream being "caught" is terminated, or a later stream merged with the stream under consideration before it could effect a merge).

The next section describes experiments conducted with the prototype. The goal of the experimentation was to determine the correctness of the implementation. In future work, a detailed experimental study with the prototype will be conducted.

3.5.4 Experiments

Initial experiments were conducted with both server and client components on the same machine while accessing media content from the Internet. The same media file was accessed using different instantiations of the media player (on the same machine). Experiments were also conducted in a LAN environment by collaborators at the University of Wisconsin. The results indicate that the protocol is working correctly.

Obtaining accounts on remote sites with multicast capability has been difficult, and so far access has been available only to machines at the University of Calgary and the University of Wisconsin. This somewhat limited the ability to test the implementation exhaustively in the wide-area context. Several experiments were conducted between August and October 2003, in which the server was located either at Wisconsin or at Calgary, with clients located at one or both locations. These experiments have further demonstrated that the prototype is indeed working correctly.

3.6 Summary

Two new protocols are developed, namely Reliable Periodic Broadcast (RPB) and Reliable Bandwidth Skimming (RBS), which effectively address the problem of scalable and reliable on-demand media streaming. The design and analysis of these protocols are, in part, guided by the new lower bounds developed on the required server bandwidth for both immediate service protocols and protocols that have a bounded start-up delay, assuming that the maximum packet loss rate is p. Performance evaluation of the protocols shows that both RPB and RBS can nearly achieve the applicable lower bounds, and therefore can achieve nearly the best possible scalability. To demonstrate implementation feasibility, the RBS protocol has been incorporated in the SWORD prototype.

Chapter 4

Rate Control

This chapter investigates multicast congestion control policies for scalable streaming applications. A desirable property of these applications is that the reception rate of each client be determined in accordance with the characteristics of the transmission path from the server to the client. Among clients participating in a multicast session, however, there may be substantial heterogeneity with respect to loss rates, delays, and available bandwidth. To accommodate this heterogeneity, the server may transmit data for the same media object using multiple multicast channels, thus allowing clients to adjust their reception rates by joining/leaving some of these channels. Congestion control protocols that adhere to this general approach are known as *multirate* congestion control protocols.

Multirate congestion control protocols can be applied in different ways, depending on the particular application. Live or scheduled broadcast applications, for example, can use cumulative layered encoding, in which a base layer and a small number of enhancement layers are each broadcast on a different multicast channel. Each client receives the base layer and a number of enhancement layers depending on its target reception rate. This layered transmission scheme can also be used for on-demand streaming using a periodic broadcast protocol such as Reliable Periodic Broadcast [84], in which the media object is partitioned into segments that are periodically multicast according to a predetermined transmission schedule. The congestion control policy determines the target reception rate of each client. For a given target reception rate, a client will determine which segment/layer combinations to receive, based on a policy that attempts to maximizes the playback quality at the client. Such a policy is proposed in the next chapter. In the context of bandwidth skimming protocols [41], the target reception rate of a client could determine the number of "catch up" streams it receives. Note that the granularity of the available rate adjustments may be quite coarse due to limitations imposed by the media encoding and/or the scalable protocol design. In general, the granularity of rate adjustments depends on the rate of transmission on each multicast channel.

Several multirate congestion control protocols have been proposed [88, 125, 126, 74, 45, 119, 23, 82]. The proposals most relevant to this work were reviewed in Section 2.4.4, including Fair Layer Increase/Decrease with Dynamic Layering (FLID-DL) [22], Packet pair receiver driven Layered Multicast (PLM) [74], and Wave and Equation Based Rate Control (WEBRC) [82]. The FLID-DL protocol uses a TCP Reno throughput model [93] with a fixed value for the round-trip time (RTT) variable to determine the placement of synchronization points in the data stream. These synchronization points control when a client may join another multicast channel. Clients leave a multicast channel if packet losses are observed in the previous observation interval; otherwise, on receipt of an enabling synchronization point, a multicast join is performed. The behavior of this protocol is similar to TCP Reno in that the target reception rate decreases only when packet loss occurs and the reception rate exhibits additive increase, multiplicative decrease (AIMD) behavior.

The WEBRC protocol is an *equation-based* multirate congestion control protocol in which each client independently uses a TCP Reno throughput model to explicitly determine its bandwidth share. A key characteristic of WEBRC is that the transmission on each channel is in *waves*, that is, at the start of a cycle the transmission rate increases to a maximum value, and then exponentially decreases over time with transmission finally ceasing until the beginning of the next cycle. Depending when channels are subscribed to, fine-grained rate changes are possible. This protocol is suitable for bulk download applications, but it is not clear how the protocol would be applied with various types of streaming applications. For example, layered video streaming using WEBRC may be possible with each layer being transmitted during a particular time period in each channels transmission cycle, but clients may receive partial layers owing to the fine-grained rate adaptation, that may not be fruitfully used. Furthermore, it is not clear *a priori* how the wave-like transmission structure could be applied with applications using bandwidth skimming protocols.

The congestion control mechanism of TCP Reno (and similar variants such as Newreno and Sack) induces packet loss, and therefore, multirate congestion control protocols that behave similarly will also induce packet loss. An alternative to TCP Reno-like congestion control is the approach used in the PLM protocol. In this protocol, the server sends packets in pairs, and receivers use the spacing between the two packets to estimate the bandwidth on the path between the server and the client. PLM's bandwidth inference process does not induce packet loss, clients fairly share bandwidth with TCP, and also quickly converge to their respective fair share bandwidths. This approach, however, has the disadvantage of making the assumption that routers on the path between the server and the client implement a fair queuing scheduling discipline.

The work in this chapter has as its goal a multirate congestion control protocol that provides less oscillatory reception rates, shares bandwidth fairly with TCP and other rate controlled flows over a wide range of network conditions, and operates without necessarily inducing packet losses while probing for bandwidth. These goals are largely achieved in the proposed Vegas Multicast Rate Control (VMRC) protocol. This protocol is an equation-based multirate congestion control protocol that employs a recently proposed TCP Vegas throughput model [112]. A VMRC sender transmits data using multiple multicast channels. Each VMRC client measures loss rate and queuing delays, and computes a target reception rate that determines how many multicast channels it may receive. The VMRC protocol exhibits TCP Vegas-like behavior and has one key advantage compared to protocols with TCP Reno-like behavior, namely the protocol operates without inducing packet losses when the bottleneck link is lightly loaded.

The previously defined TCP Vegas protocol includes two static threshold parameters (α and β) that determine the target number of packets queued in the network for a TCP Vegas flow. Depending on the values of these parameters, the previously defined Vegas protocol is either too aggressive or too meek when operating in a network that also has some number of TCP Reno flows [112]. The VMRC protocol incorporates a new technique for dynamically varying the Vegas threshold parameters based on the measured characteristics of the network. This technique allows a TCP Vegas based rate control flow to share bandwidth fairly with other types of flows, including widely deployed versions of TCP such as TCP Reno. Thus, the technique might be fruitfully incorporated in TCP Vegas itself to aid in its incremental deployment.

An extensive performance evaluation study of VMRC is presented which demonstrates the performance properties of the protocol. To illustrate the benefits of Vegas-like congestion control, the performance of VMRC is compared with that of an analogous protocol that is based on a TCP Reno throughput model. Furthermore, the design of VMRC is evaluated along the key dimensions of synchronization policy, RTT and propagation delay measurement techniques, data transmission policy, and protocol reactivity. The main findings of this performance study are:

- Coherency among clients behind a common bottleneck link can be attained by using common protocol mechanisms that depend only on channel history and not on information known only locally, rather than explicit synchronization schemes.

- Explicitly tracking RTT and using the measured values in the Vegas or Reno throughput model may be undesirable as clients behind the same bottleneck link may compute different target reception rates such that their channel subscriptions differ, thus yielding inefficient use of multicast transmissions. Experiments show that estimating RTT as the sum of a delay component that reflects changes in queuing along the transmission path and a fixed value (that determines the aggressiveness of the protocol) can yield good fairness and reactivity compared to protocols that use a fixed value for the RTT parameter.

- The packet loss rate experienced by a VMRC session typically increases with an increase in the burstiness of the data transmission scheme. Bursty transmissions, however, result in faster convergence of clients to their optimal bandwidth share. The transmission scheme appears to have minimal impact on TCP fairness.

- Clients need to explicitly track both short-term and long-term averages of loss rate and delay measures to attain both good reactivity and good stability.

The remainder of this chapter is organized as follows. Section 4.1 presents a brief overview of equation-based rate control. The VMRC protocol is described in Section 4.2. An experimental evaluation of VMRC is presented in Section 4.3. The chapter is summarized in Section 4.4.

4.1 Overview of Equation-Based Rate Control

Multimedia flows that use UDP over shared transmission paths in best effort networks such as the Internet must avoid causing uncontrolled congestion. One recommended approach is that the bandwidth usage of a multimedia flow should not exceed that of a TCP connection under similar conditions [19]. A multimedia flow that satisfies this criterion is referred to as "TCP-friendly" or "TCP-compatible", and has the advantage of sharing bandwidth reasonably fairly with the large volume of traffic that is predominantly TCP.

The design of TCP-friendly rate control protocols for multimedia flows has received considerable attention in recent years. Of interest in this thesis is the *equation-based* approach [125, 45, 119, 82], in which the reception rate at each client is determined by direct application of a TCP throughput model. This approach offers the potential of sharing bandwidth fairly with TCP flows while achieving lower variation in reception rates than with a TCP-like increase/decrease approach, in which the reception rate is abruptly reduced in response to each loss event.

Prior proposals for equation-based rate control have utilized a TCP Reno throughput equation. This thesis presents and evaluates a congestion control protocol based on a TCP Vegas throughput model. The Reno and Vegas throughput models are reviewed in Sections 4.1.1 and 4.1.2, respectively. Techniques for online estimation of model parameters are briefly discussed in Section 4.1.3.

4.1.1 Reno Throughput Model

Models have been proposed that predict the steady state throughput of a long duration TCP Reno flow as a function of the average RTT and the probability of packet loss [87, 93, 57]. The TCP Reno throughput equation used in this work as well as in many prior protocols is

$$\Lambda_{reno} = \frac{1}{R\sqrt{\frac{2p}{3}} + TO\left(3\sqrt{\frac{3p}{8}}\right) p\left(1+32p^2\right)}, \tag{4.1}$$

where Λ_{reno} specifies the throughput in packets per second, R is the average RTT in seconds, TO is the TCP retransmission timeout value, and p is the steady state loss event rate [93].

Designing control algorithms for closed loop feedback systems is difficult since the inputs to the control equations may change based on the action taken for a given set of inputs (e.g., the measured loss event rate of a flow exhibits dependencies on the sending rate of the flow). This observation motivated revisiting the TCP Reno throughput model outlined above, and developing an alternative model that measures loss rate in units of time between loss events (in the hope that this measure will be independent of the reception rate of the receiver), instead of packets between loss events. This model is discussed in Appendix A.

4.1.2 Vegas Throughput Model

In recent work, a throughput model for TCP Vegas has been developed [112]. Unlike TCP Reno, which induces losses to estimate available bandwidth, the TCP Vegas congestion control mechanism attempts to detect congestion in the network before packet loss occurs. Vegas uses increases in queuing delay to detect congestion and attempts to maintain, on average, between α and β unacknowledged packets queued in the network, where α and β are the "threshold" parameters of the protocol. If packet losses are negligible, the throughput of a Vegas flow is given as

$$\Lambda_{vegas}^{no-loss} = \frac{\beta}{R - baseRTT} = \frac{\beta}{\Delta}, \tag{4.2}$$

where $\Lambda_{vegas}^{no-loss}$ specifies the throughput in packets per second, R is the average RTT, and $baseRTT$ is the minimum observed RTT for the flow [112].

The throughput of a Vegas flow does not have any RTT bias when packet losses are negligible, in the sense that it depends *only* on the average queuing delay observed along the path from the sender to the receiver. When packet losses and timeouts are experienced, the throughput of a Vegas flow is given by

$$\Lambda_{vegas}^{loss} = \frac{(n+1)(\frac{1-p}{p} + W)}{NR + TO}, \tag{4.3}$$

where $W = \frac{R(\alpha+\beta)}{2\Delta}$, $n = \frac{1-p_{to}}{p_{to}}$, $p_{to} = \min(1, \frac{p+p(1-p)(1-(1-p)^{W-1})}{1-(1-p)^W})$, and
$N = 2logW + (n+1)\frac{1-p}{pW} + (1+\frac{n}{4})\frac{W}{8} + \frac{9n}{8} - \frac{11}{4} + \frac{4-2^{logW}}{W}$.
When packet losses are observed, Vegas flows exhibit some RTT bias, but the bias is lower than that of Reno flows (e.g., see [90, 112]).

Simulation studies have also shown that the relative aggressiveness of Vegas flows depends on the threshold parameters and the buffer space available at the bottleneck queue [112]. Choosing a threshold parameter that allows a Vegas flow to compete fairly with other Vegas flows as well as flows employing more aggressive TCP variants such as Reno is a hard problem that has been an impediment to deployment of Vegas.

4.1.3 Online Estimate of RTT and Loss Event Rate

The Exponential Weighted Moving Average (EWMA) technique is widely used for obtaining smooth estimates of RTT. This technique assigns a weight α to the most recent estimate of average RTT (R) and a weight of $(1 - \alpha)$ to the most recent measure of RTT (R_t) to obtain a new estimate R as follows:

$$R = \alpha R + (1 - \alpha) R_t. \tag{4.4}$$

With EWMA the weight given to an RTT measurement decays exponentially with the age of that measurement.

The Average Loss Interval (ALI) method has been widely used for obtaining online estimates of loss event rates [45]. Let s_i denote the number of packets received in the i^{th} most recent loss interval (i.e., period of time between loss events), and s_0 be the number of packets received since the most recent loss event. The average loss interval length $\bar{s}$ (measured in number of packets) is calculated as a weighted average over a window of n loss events as:

$$\bar{s} = \frac{\sum_{i=1}^{n} w_i s_i}{\sum_{i=1}^{n} w_i}. \tag{4.5}$$

The weights w_i are chosen such that the most recent loss intervals receive equal weight, with the weights of older intervals gradually decreasing to zero. In [45], $n = 8$ is used, with the following weights: $1, 1, 1, 1, 0.8, 0.6, 0.4, 0.2$.

The above calculation is repeated using s_0 through s_{n-1} to compute $\bar{s}_{new}$. The average loss interval length is taken as $\max(\bar{s}, \bar{s}_{new})$, thus including the most recent loss interval only when it reduces the estimated loss rate. The corresponding loss event rate p is simply the inverse of the average loss interval length.

An important issue with both ALI and EWMA is determining how much weight should be given to the most recent samples. Better noise attenuation can be achieved by using a high value for the constant α in EWMA and by using a large n (and/or more balanced weights) in ALI, but at the cost of diminished reactivity. Placing greater weight on more recent samples may, however, result in oscillatory throughput estimates when the resulting values are used in TCP Reno or Vegas throughput models.

Table 4.1: VMRC Protocol Notation

Description	**Name**	**Default Value**
Protocol invocation granularity (time slot duration)	τ	0.1 seconds
Minimum separation between join experiments	T_{add}	20τ
Threshold for quick drop	T_{drop}	20τ
Start-up phase duration	$T_{startup}$	50τ
Short-term average RTT	R_{st}	
Long-term average RTT	R_{lt}	
Short-term average queuing delay	Δ_{st}	
Long-term average queuing delay	Δ_{lt}	
Short-term average loss event rate	p_{st}	
Long-term average loss event rate	p_{lt}	
Average reception rate in a time slot	B_{recp}	
Expected reception rate	B_{curr}	
Add hysteresis parameter	γ_1	0.5
Drop hysteresis parameter	γ_2	0.5
Minimum Vegas threshold parameter	β_{min}	
Maximum Vegas threshold parameter	β_{max}	

4.2 The VMRC Protocol

VMRC is an end-to-end congestion control protocol that does not require any special support from the network other than availability of application or IP multicast. The protocol assumes that the server transmits data for the same media object using multiple multicast channels. The description of the protocol given here assumes a cumulative layered encoding. Each client subscribes to a number of multicast channels with an associated rate approximately equal to that determined by using the Vegas throughput equations described in Section 4.1.2. The subsequent sections describe components of the protocol in detail. Notation used in the protocol description is summarized in Table 4.1.

4.2.1 Protocol Invocation Granularity

The protocol invocation granularity determines how often clients compute new throughput estimates and take appropriate channel add/drop decisions. Receivers invoke the protocol once every τ seconds. The time between two invocation points will be referred to as a *time slot*. Time slots should be long enough to allow obtaining reliable estimates of the throughput model parameters. However, even with reliable estimates, a channel addition may turn out to be be too aggressive, due to the coarse granularity at which rate changes are possible. This motivates smaller time slots, which could permit quicker channel drops. Very small time slots, however, may not allow enough time following a channel drop for packets that are back logged in the bottleneck queue to drain, and may result in an additional unnecessary channel drop. To accommodate these conflicting objectives the default value of τ is set to 0.1 seconds, and the add operations of the protocol depend on the more stable long-term averages of the throughput model parameters. The protocol allows quick drops following a recent add, based on short-term averages of the parameters.

4.2.2 Subscription Change Points

Subscription change points control when channel add/drop operations may be performed. Two considerations can be identified, namely: 1) how frequently should add/drop operations be allowed; and 2) should the add/drop operations of multiple clients be coordinated.

Note that since only coarse-grained reception rate changes are possible, only an approximate match can be achieved between the target and actual reception rates. Allowing frequent channel additions may result in oscillation. Oscillating channel subscription is undesirable as it serves only to increase congestion in the bottleneck path. Therefore, to reduce oscillations and control the amount of congestion being introduced, the protocol allows channel additions once every T_{add} seconds. Channel drops are allowed to take place as quickly as possible (i.e., on any time slot boundary) to prevent causing long periods of congestion. The default value for T_{add} is 20τ seconds.

Coordinating channel additions by clients behind a common bottleneck may be important since addition of an unsustainable channel by one client may cause packet loss, and in turn cause other clients behind the same bottleneck link to drop a channel. The ability to quickly drop a channel, however, does not impede the convergence of other clients behind a bottleneck link to their optimal subscription level. As the number of clients in a session increases, the problem of uncoordinated channel additions exacerbates because of the increased amount of congestion being introduced in the network. Therefore, many prior protocols have proposed mechanisms to carefully orchestrate channel additions [88, 126, 22].

The VMRC protocol uses a *weak synchronization* policy to coordinate channel additions. In this policy, the server divides the data stream into fixed length time slots. Packet headers include a time slot index that advances by one when a new time slot begins. Clients may attempt to add *any* channel on the receipt of a packet from a new time slot. To accommodate the constraint on frequency of channel additions discussed above, add experiments are allowed at the beginning of a new time slot provided the time slot index modulo T_{add} is zero. An alternative to this approach is the *no synchronization* policy, where each client delimits time slots independently. Experimental evaluation shows that there are some potential benefits to using weak synchronization.

4.2.3 Defining RTT and Related Parameters

The RTT and related parameters required by the Vegas throughput model are R, TO, and Δ, where R is the average RTT, TO is the retransmission timeout duration typically approximated as $4R$, and Δ is the average increase in the one-way queuing delay.

Assuming that the server timestamps each packet, the one-way transmission time (OTT) from the server to the client can be obtained by taking the difference between the time at which the packet was sent (T_{server}) and the time when the packet was received (T_{client}), as measured at the client. Using the OTT measurements, a client can maintain an estimate of the average OTT (O), and determine the minimum OTT ($baseOTT$) to compute Δ as follows:

$$\Delta = O - baseOTT. \tag{4.6}$$

Note that errors due to lack of clock synchronization cancel out when the minimum observed OTT is subtracted from the current estimate of the average OTT. Prior research has observed that clock drifts on a time scale of minutes to hours can occur [97, 91]. For long duration sessions, this problem could be addressed by infrequently updating $baseOTT$ (e.g., by computing $baseOTT$ over a moving window of some duration).

With multiple clients, each client may have a different RTT to the server, and thus using actual RTT measure in the throughput model may lead to clients behind a common bottleneck link achieving different subscription levels. Furthermore, for the content delivery applications that are the focus of this work, the application has little or no reverse traffic, and thus additional mechanisms may be needed to measure RTT even in the context of a single client. In this work, three alternative techniques for RTT estimation are considered. One approach is to obtain a variable RTT estimate that is the sum of the OTT plus the the return path propagation delay. A second option is to use a fixed constant value for the RTT of all clients. A third approach is to estimate RTT as the sum of an estimate of the transmission path queuing delay plus a fixed value. Elaborations on each of these techniques is provided next.

In the first approach mentioned above, each client performs an end-to-end control information exchange with the server at start-up, which allows it to obtain an initial RTT estimate ($R_{initial}$), and an initial value for OTT between the source and the client ($O_{initial}$). On each data packet reception, an RTT estimate R_t can be obtained as follows:

$$R_t = R_{initial} + \left[(T_{client} - T_{server}) - O_{initial} \right]. \tag{4.7}$$

This technique tracks the RTT between a server and client as would be seen by a TCP flow traversing the same path, assuming that most of the variability in RTT is owing to congestion in the forward path. Although this technique may yield more TCP-friendly behavior, different clients behind a common bottleneck link may potentially receive different channel subscriptions. Also, there may be scalability problems with a large number of clients attempting end-to-end control information exchange with the server.

The second option is to use a fixed value as the RTT parameter, identical for all clients. This approach is adopted by FLID-DL. An advantage of this approach is that different clients that are behind the same bottleneck link but with substantially different round-trip propagation delays, could compute the same target reception rate and thus receive the same channel subscription, which is desirable. Note that even with this option, with the Vegas model, clients can react to changes in the queuing delay along the transmission path of the multicast packets, as this is reflected in the measured Δ parameter described above.

The third approach, which is the default for VMRC, estimates RTT as the sum of a fixed value and Δ, where Δ is measured as described above. Compared to using a fixed value as the RTT estimate, this technique is more responsive, and at the same time, enables clients behind the same bottleneck link with different round-trip propagation delays to obtain similar estimates of RTT.

4.2.4 Defining Loss Events

The VMRC protocol measures the rate at which loss events occur using the ALI technique. The reason for measuring the loss event rate as opposed to the fraction of packets lost is as follows. First, various TCP variants generally reduce their congestion window, only once, in response to several losses in a congestion window. Second, with drop-tail queuing at the routers, congestion can significantly increase the probability of losing several packets within a congestion window.

Careful consideration of what constitutes one loss event is necessary to use the ALI technique. VMRC assumes that a new loss event occurs when the time between two packet loss incidents exceeds at least by the length of a time slot. The loss interval is defined as the period of time from the last packet loss of one loss event, until the first packet loss of the next event. Note that any packets received in between losses in a loss event are not included in the calculation of the length (as measured in numbers of packets) of a loss interval.

4.2.5 Long-Term and Short-Term Parameters

VMRC maintains both short-term and long-term averages of the queuing delay, RTT, and loss event rate, for use as the R, Δ, and p parameters. The long-term estimates are reliable measures that mainly guide channel additions. The short-term measures enable quick detection of channel additions that are too aggressive and also prevent channel addition when new congestion is detected.

The short-term average R_{st} of the RTT estimate is updated on receipt of a packet and is an EWMA with the most recent sample R_t given a high weight (e.g., 25%). The long-term average R_{lt} of the RTT estimates is updated at the beginning of a new time slot and is an EWMA of R_{st} with a small weight (e.g., 5%) assigned to the most recent value. The short-term average O_{st} and the long-term average O_{lt} of the OTT estimate is obtained in a similar fashion from the most recent OTT sample. The short-term average Δ_{st} queuing delay component is calculated as $O_{st} - baseOTT$, while the long-term average O_{lt} queuing delay is calculated as $O_{lt} - baseOTT$.

The long-term average p_{lt} of the loss event rate is a ALI measure with $n = 8$. The short-term average p_{st} of the loss event rate, is the inverse of the number of packets received in the most recent loss free interval, s_0. When $s_0 = 0$, $p_{st} = 1$.

4.2.6 Dynamic Values of TCP Vegas Thresholds

From the Vegas throughput model, it is evident that the bandwidth obtained by a TCP Vegas flow depends on the choice of the threshold parameters α and β. Simulation studies in the literature have demonstrated that depending on the choice of α and β, a Vegas flow can be more or less aggressive than a competing TCP flow. This work proposes a technique for dynamically setting the Vegas threshold parameters based on whether the flow reaches *stable backlog* or not [112]. Stable backlog means that the Vegas flow attains the desired congestion window of W between loss events.

Note that recent work has shown that both threshold parameters can be set to the same value, and this assumption is made here as well [112]. The condition for stable backlog to be attainable is approximated by the following [112]:

$$p \leq \frac{32}{7W^2 - 36W + 32}. \tag{4.8}$$

At start-up, β is initialized to a parameter β_{min}. At the beginning of each new time slot, the above is solved for W (by taking the expression as an equality) using the long-term average p_{lt} of the loss event rate. The computed value of W is then used to derive β_{new} as $(W\Delta_{lt})/R_{lt}$, and β is updated using an EWMA with a small weight (e.g., 5%) assigned to β_{new}. If the loss rate is negligible, β_{new} is set to a parameter β_{max}. Parameters β_{min} and β_{max} define the possible range of aggressiveness of the protocol and are set to allow a wide range of aggressiveness. The experiments in this chapter set β_{min} and β_{max} to the minimum and maximum number of packets transmitted by the server in a single time slot as determined by the minimum and maximum allowed subscription levels, respectively.

4.2.7 Steady State Add/Drop Rules

In steady state, clients estimate their bandwidth share Λ_{est} by applying the throughput model, and take decisions to increase, decrease, or maintain the current level of subscription. Using parameters based on the long-term and short-term averages discussed earlier, a client can compute corresponding long-term and short-term estimates Λ_{st} and Λ_{lt} of the fair share bandwidth.

Let B_i denote the bandwidth required to support the currently subscribed i multicast channels. A client can increase its subscription to $i+1$ if $\min[\Lambda_{st}, \Lambda_{lt}] \geq B_{i+1} + \gamma_2(B_{i+1} - B_i)$. Using the minimum of the long-term and short-term bandwidth estimates allows channel additions only when there is a high probability of the addition being successful. That is, the long-term estimates indicate that there is available bandwidth and the short-term estimate confirms that no new congestion has been witnessed recently. The hysteresis parameters γ_1 and γ_2 are used to decrease or eliminate oscillations in the subscription level.

The rule for decreasing the subscription level is designed to allow early channel drops following a channel addition that is deemed unsustainable, and/or if the network parameters reflect persistent congestion. Ascertaining whether or not an increase in the subscription level results in congestion is particularly important due to the coarse granularity of the possible bandwidth changes. Therefore, if the client added a new channel in the recent T_{drop} time slots without an intervening layer drop, the subscription level is dropped to $i-1$ if $\min[\Lambda_{st}, \Lambda_{lt}] < B_i - \gamma_1(B_i - B_{i-1})$; otherwise, the condition for dropping a channel is $\max[\Lambda_{st}, \Lambda_{lt}] < B_i - \gamma_1(B_i - B_{i-1})$.

An important design criterion for the add/drop rules is that all clients behind the same bottleneck link should use the same information to determine their layer subscriptions, to the extent possible, particularly with respect to layer additions, and to drops of layers that have not been recently added. For example, it is conceivable that to stabilize a client's subscription level and to allow time for queues to drain, a client could apply a *dead period* [126], during which no subscription changes occur. Such mechanisms, however, can impede and even prohibit coherency among clients, as all clients behind a common bottleneck link can not be guaranteed to invoke a dead period at the same time.

4.2.8 Start-up Add/Drop Rules

When a client joins a multicast session, it starts listening to the base layer. Since reliable estimates of RTT, queuing delay, and loss event rate are not available in the start-up phase, the throughput model can not be applied, and the client enters a slow start phase. During slow start, a client increases its subscription level by joining one multicast channel per time slot provided the previous join has succeeded (and not when network queuing is increasing). Thus, an additional condition for a join is that the average rate of reception in the previous time slot, B_{recp}, is at least 95% of the current subscription rate, B_{curr}.

Slow start is terminated when congestion is detected. One signal of congestion is packet loss. With large network queues, unsustainable subscriptions may not immediately cause a packet loss. However, if B_{recp} is less than $0.95B_{curr}$ for one time slot since the join has been successful, and therefore, indicating queue buildup, slow start is terminated. On terminating slow start, the reception rate is reduced by dropping the highest subscribed multicast channel. When slow start is terminated, a reliable estimate of loss event rate is not available, and, therefore, Equation 4.2 is used to obtain the throughput estimates until losses occur.

Recall that during steady state, a client may attempt channel additions once every T_{add} time slots. Thus, if the slow start phase is terminated before a client reaches its optimal subscription level, it may encounter a long delay to reach the optimal subscription. To alleviate this potential problem, during the first 50τ seconds after a client has joined the session, the above restriction is not enforced.

4.3 Protocol Performance

This section presents a simulation study of the VMRC protocol. The simulations evaluate the performance of the protocol under a variety of network conditions. The specific goals of the performance study are to explore the performance properties of VMRC and to illustrate the benefits of Vegas-like rate control over Reno-like rate control. The implementation of the VMRC and the analogous Reno throughput model based protocol is discussed in Section 4.3.1. Section 4.3.2 discusses the simulation topology and the models of background traffic considered in this study. The performance metrics are defined in Section 4.3.3. Experimental evaluation of VMRC and the equivalent Reno throughput model based protocol is presented in Section 4.3.4, followed by an evaluation of the design choices available in VMRC in Section 4.3.5.

4.3.1 Implementation

As mentioned above, the performance of VMRC is compared to that of an analogous protocol based on the Reno throughput model. This protocol is referred to here as

Reno Multicast Rate Control (RMRC) in this work. RMRC uses the same protocol components as VMRC except for the throughput model used and the manner in which loss event rate is seeded when clients exit slow start. Recall that on terminating slow start, VMRC clients keep using Equation 4.2 until losses occur and an estimate of the loss event rate is available. The RMRC protocol, however, requires an estimate of the loss event rate immediately upon termination of slow start so that the throughput model can be used. Therefore, upon terminating slow start, an estimate of the loss event rate is calculated by solving the Reno throughput equation using the current rate of reception and the current estimate of RTT. This estimated loss event rate is used to seed the ALI loss history.

Both protocols were implemented in the *ns-2* network simulator [103]. The performance evaluation study considers a cumulative layered video multicast application, where clients always receive the base layer, and some number of additional enhancement layers. The experiments reported here assume a base rate $B_0 = 256$ kbps, and a layer multiplicative factor $c = 1.5$ (i.e., cumulative transmission rate of i layers is $B_i = B_0 c^{i-1}$). A total of 9 layers are considered, thus allowing the following reception rates: 256, 384, 576, 864, 1296, 1944, 2916, 4374, and 6561 Kbps.

Timer limitations of operating systems can guarantee only a coarse granularity for scheduling packets (e.g., 50 to 100 milliseconds). Such coarse-grained packet scheduling policies also reduce system overhead. Evidence from empirical studies of multimedia audio/video traces confirm that multimedia streams are indeed bursty on small time scales, although the overall bit rate is constant at longer time scales [89, 70]. The *ns-2* implementations of the multicast server models this observed burstiness by scheduling packet transmissions once every δ milliseconds. For each layer, the amount of data to be transmitted in the time interval δ is used to determine how many data packets must be transmitted. A maximum packet size of 1 KB is specified, but the sender may have to transmit smaller sized packets to account for the data accumulated since the last packet scheduling event. The default value for δ is 100 milliseconds. In Section 4.3.5, the influence of bursty transmissions is also investigated.

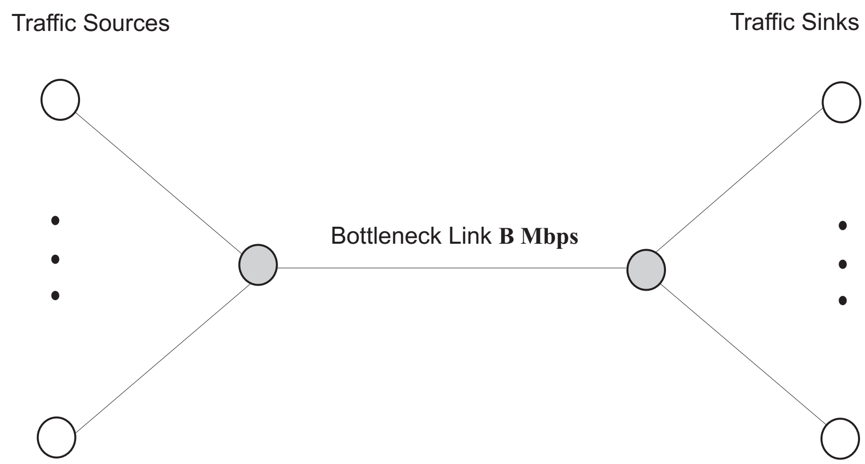


Figure 4.1: Simulation Topology

4.3.2 Network Topology and Background Traffic

The simulations reported here consider a simple dumbbell network topology as shown in Figure 4.1, where there is a single common bottleneck link between all sources and sinks. Each source/sink is connected to the bottleneck link via a high bandwidth access link. The propagation delays of the access links is varied to simulate the desired round-trip propagation delay between a source/sink pair. The routers use FIFO queuing with drop-tail queue management. The buffer capacity of the bottleneck queue is large compared to the bandwidth-delay product of the bottleneck link (e.g., 5 to 10 times the bandwidth-delay product of the bottleneck link). This simple network topology clearly does not capture the complexity of the Internet, but allows highlighting the main aspects of the protocol. Following the recommendations in [99, 51], this experimental study utilizes a rich mix of competing background traffic in this simple topology to encompass a wide spectrum of the protocol's performance.

Congestion in the Internet can occur at low bandwidth, low multiplexing environments such as access links or at high bandwidth, high multiplexing environments such as backbone links. Therefore, to cover a range of possible current and future network conditions, the experiments vary both the link capacity and the degree/type of background traffic. Bottlenecks closer to the client are simulated using low bandwidth bottleneck links (e.g., ≤ 6 Mbps) with little or no background traffic. Bottlenecks in the backbone are simulated using high bandwidth bottleneck links (e.g., ≥ 10 Mbps) with varying degrees of background load simulated using a mix of long duration FTP and short duration ON/OFF HTTP sessions.

The HTTP sessions are simulated using a model similar to that used in [81, 112]. Each HTTP client sends requests to a dedicated HTTP server. Clients send a single packet request to the associated server, and upon receiving the request, the server sends a file to the client. After a file transfer is complete, the client generates another request to the server after a think time that is exponentially distributed with a mean of 500 milliseconds. To simulate the observed heavy-tailed nature of HTTP transfers (e.g., [11, 83, 85]), the file sizes are drawn from a Pareto distribution with mean 48 KB and shape 1.2.

Both HTTP and FTP session use the Newreno variant of TCP with a maximum congestion window size of 64 packets. The maximum packet size is 1 KB. The background traffic in the experiments consider round-trip propagation delays between 20 and 460 milliseconds [6, 66, 51, 112].

In simulations where multiple long duration flows share a single bottleneck link, systematic discrimination has been observed against some connections [47]. Such *phase effects* rarely occur in the real Internet and are mostly an artifact of unrealistic simulation scenarios (e.g., multiple long duration flows, same size packets, no traffic in the reverse direction, etc.) [51]. Simulations that consider a mix of long and short duration flows, with different round-trip propagation delays (such as the ones presented in this study), however, seldom experience this problem. As a precautionary measure to avoid phase effects, the following two steps were taken. First, all flows were started at slightly different times. Second, a small random delay between transmission of TCP packets as well as between transmission of bursts of multicast packets was also introduced.

The *ns-2* simulator does not model multicast join/leave latencies as observed for the currently deployed protocols in the Internet. Specifically, the joins/leaves are instantaneous in the simulations. Note that this does not impact the qualitative comparison between VMRC and RMRC presented here, as both multicast protocols are expected to be similarly influenced. Also, it may be reasonable to expect that these problems may be addressed if multicast were to become popular. Finally, it should be mentioned that the protocols presented here could use techniques such as the Dynamic Layering scheme of FLID-DL to address large multicast leave latencies.

4.3.3 Evaluation Metrics

A multirate congestion control protocol for streaming media can be assessed on the basis of the following five characteristics: TCP fairness, smoothness of the reception rate, session loss characteristics, coherency among clients behind a common bottleneck link, and the responsiveness of the protocol to sudden changes in the network conditions. Thus, this study utilizes the following performance metrics:

- Convergence time: For a group of clients behind a common bottleneck link that join a multicast session at times close to each other, convergence time is measured as the difference between the time at which all clients reach the optimal subscription level (and stay at this level except for the occasional join experiments) and the time at which the last client joined the session. Longer convergence times reflect inefficient use of the available bandwidth.

- Transient loss rate: VMRC allows clients to join multicast channels once every time slot when operating in the start-up phase. This allows clients to quickly attain the optimal subscription level. For a group of clients that join a multicast session at times close to each other, transient loss rate is defined as the fraction of packets lost until all clients of the session exit their start-up phase. Transient loss quantifies the impact of a new client joining an ongoing multicast session.

- Steady state loss rate: This is the fraction of packets lost by a multicast session, excluding the initial startup period. This metric, together with the achieved layer subscription, reflects the perceived quality of the streaming media at the clients.

- Average session throughput: This is measured as the bandwidth obtained by a multicast session during the last three quarters of a simulation, and can be used to assess fairness of the multicast session.

- Session throughput over time: Average throughput is calculated over a moving window of size one second for the entire duration of the simulation run. This metric is used to judge the smoothness of the reception rate, and also to illustrate the responsiveness of the protocol to sudden changes in the network conditions.

- Session packet loss over time: The fraction of packets lost over a moving window of size one second is calculated for the entire simulation run. This metric can be used to identify how bursty the packet losses are, and also helps understand the effect these packet losses might have on the playback of the media at the clients.

Unless stated otherwise, simulation results are reported by running each experiment 10 times, with each run being made with a different seed for the random number generator. Typically, the results graph the average from the 10 runs, along with the observed minimum and maximum values (showed using an error bar centered on the mean).

4.3.4 Comparing Vegas and Reno Rate Control

This section presents sample results that illustrate the performance of VMRC and illustrate the benefits of using a Vegas-like congestion control mechanism for multicast applications. In the experiments, the sessions being monitored will be referred to as "foreground" flow, while all other traffic will be referred to as "background". The foreground flow can be a "VMRC" session, a "RMRC" session, or a "FTP" session that employs TCP Newreno. The multicast protocols employs weak synchronization, estimates RTT at the sum of a fixed value (100 milliseconds) and a variable queuing delay component, and schedules packet transmissions once every $\delta = 100$ milliseconds. The default values outlined in Table 4.1 are used for all other parameters.

4.3.4.1 No Background Traffic

Before evaluating protocol performance in the presence of background traffic, the protocol is studied in isolation. This enables identifying key protocol behavior and also verifies the correctness of the implementation.

Figure 4.2 shows the session throughput and packet loss (averaged using buckets of duration one second) for a single client multicast session experiment in which the bottleneck link has bandwidth of 3 Mbps and can buffer up to 80 packets. The propagation delay of the bottleneck link is 10 milliseconds and the round-trip propagation delay from the client to the server is 100 milliseconds. This figure shows the result obtained from a single representative simulation run. Observe that the multicast sessions quickly converge to the highest possible subscription level, and maintain this subscription for the duration of the simulation (expect for periodic join experiments). The VMRC session experiences packet loss only when slow start is terminated; otherwise, the ability of VMRC to quickly detect congestion due to a join to an unsustainable layer allows it to operates without inducing packet losses. The RMRC session, however, periodically experiences packet loss episodes during which as much as 16% of the packets transmitted over a one second interval are lost.

The next experiment studies how protocol performance scales with increase in client population. The bottleneck link is configured as in the previous experiment. The multicast session starts at time 0 and clients start at uniformly distributed random times between 0 and 5 seconds. The round-trip propagation delay of the clients is uniformly distributed between 40 and 100 milliseconds. In this experiment, the optimal subscription level consists of the base layer and six enhancement layers. Figure 4.3 plots the average, minimum and maximum convergence times, transient loss, and steady state loss from 10 simulation runs.

The convergence time depends on several factors, such as when the slow start

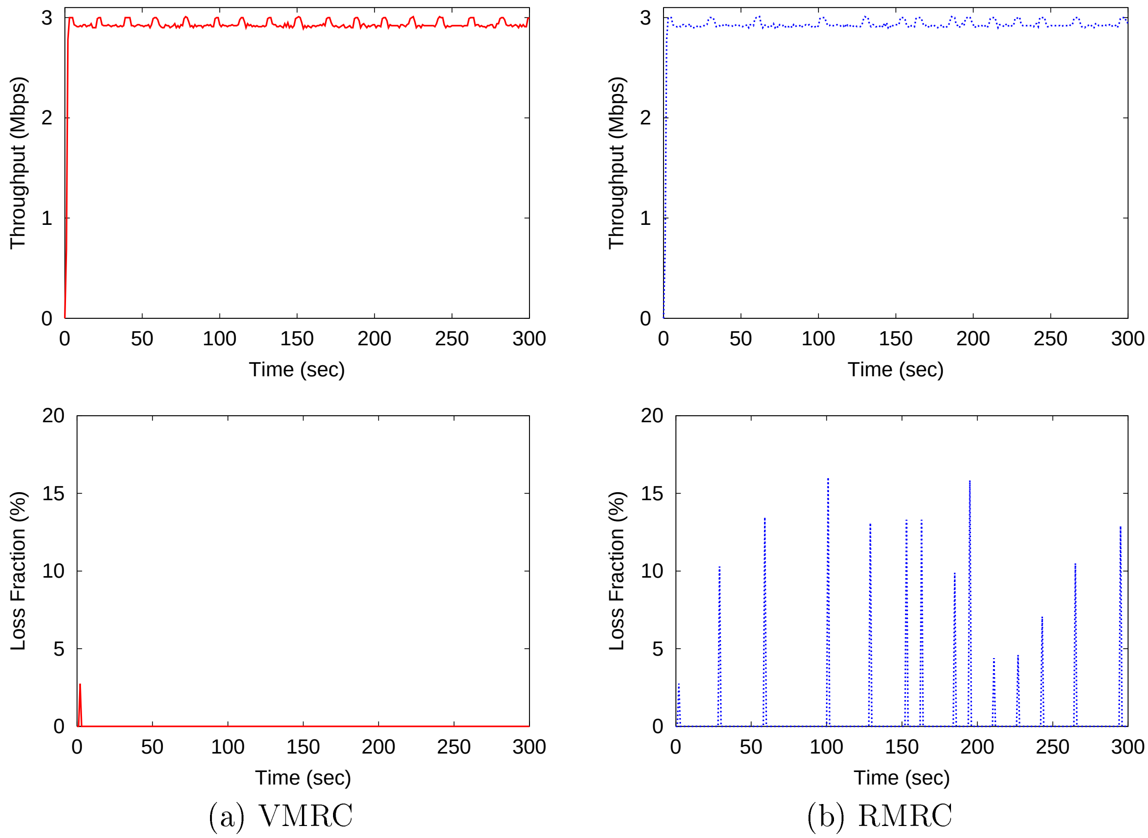
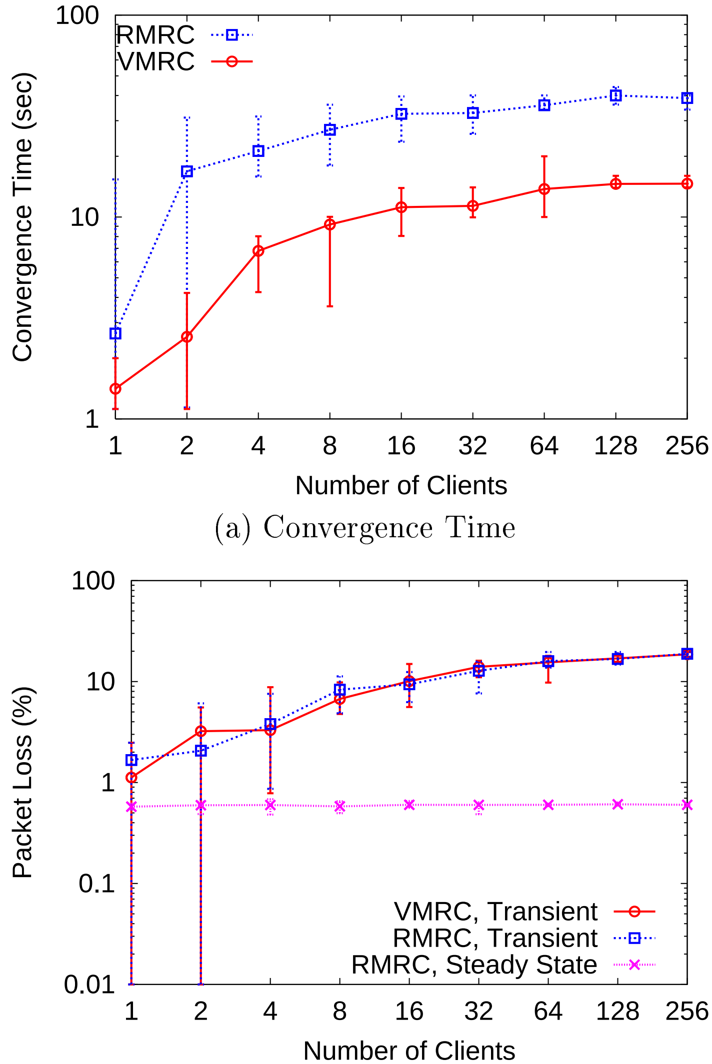


Figure 4.2: Session Throughput and Packet Loss of a Single Multicast Session (bottleneck link of capacity 3 Mbps with a drop-tail queue of 80 packets)

phase is terminated, what the estimates of the input parameters of the throughput model are at the end of slow start, and the degree of congestion being introduced in the network due to join attempts by the clients. Figure 4.3(a) shows that the convergence time of VMRC, on average, is approximately a third of RMRC. For the range of number of clients considered, the convergence times for the VMRC session is between 1 and 20 seconds, while that of the RMRC session is between 1 and 45 seconds. Faster convergence is achieved by the VMRC session because after slow start is terminated, the throughput model is applied with measured estimates available at the client, which are similar for clients behind a common bottleneck link. In contrast, on termination of slow start, RMRC needs an initial estimate of the loss event rate whose value depends on which layer the slow start phase was terminated. This initial estimate of the loss event rate may be different for different clients, and



(a) Convergence Time

(b) Packet Loss

Figure 4.3: Scalability of a Single Multicast Session with Increase in the Number of Clients (bottleneck link of capacity 3 Mbps with a drop-tail queue of 80 packets)

convergence time will depend on how quickly the loss event rate estimates of the clients converge. The ability of the Vegas throughput model to quickly detect when bandwidth is available and also detect when a recently subscribed multicast channel is unsustainable by noting the changes in the queuing delay component, also helps reduce convergence time.

In general, the convergence time is expected to increase with increase in the number of clients in the session. The results show that the increase is linear for small group size (e.g., 16 clients or less), and sublinear for larger groups. The

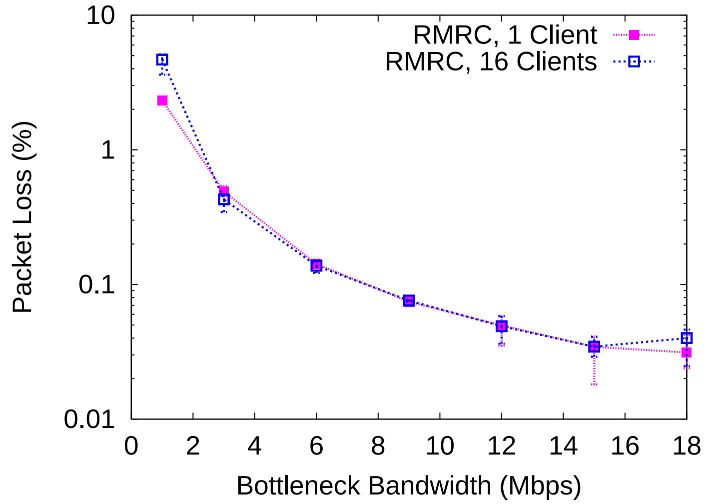


Figure 4.4: Scaling the Bottleneck Link Resources (VMRC steady state loss = 0)

difference in converge time is small for larger multicast groups as the limit of one multicast join per time slot is reached. This is because once a layer is subscribed to by one client, the subscription to that layer by another client does not change the dynamics at the bottleneck link.

From Figure 4.3(b), it can be observed that both VMRC and RMRC sessions experience similar transient packet loss. The transient loss increases exhibits the same linear/sublinear dependency on the group size as the convergence time. A key observation is that in steady state, the VMRC session experiences no packet losses, while the RMRC session experiences an average loss rate of 0.6%, reflecting the losses incurred by the periodic join experiments.

Figure 4.4 illustrates the impact of scaling bottleneck link resources on protocol performance. The initial configuration for this experiment is a bottleneck link of capacity 1 Mbps with a queue of 30 packets. At each step, the bottleneck resources are increased such that for every additional 1 Mbps of bandwidth, 30 additional packets can be queued at the bottleneck buffer. The base rate of the multicast session is also appropriately increased. The propagation delay of the bottleneck link is fixed at 10 milliseconds. The round-trip propagation delay between the server and clients is distributed between 40 and 100 milliseconds. The graphs show the average, minimum, and maximum of the percentage of packets lost in steady state for the RMRC session. Note that the percentage of packets lost decreases with increase in available bandwidth, as expected from the Reno throughput model. As expected, the VMRC session operates without losses in the steady state. Both multicast sessions reach the optimal subscription level in all the experiments.

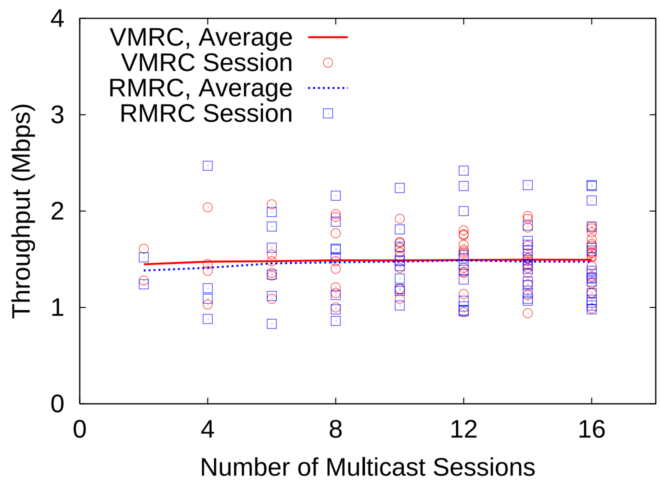
4.3.4.2 Multiple Multicast Sessions

This section evaluates how competing multicast sessions share bandwith among themselves. The initial configuration is two single client multicast sessions sharing a bottleneck link of capacity 3 Mbps with a queue of 80 packets. At each step, the number of competing sessions is increased by 2, and the bottleneck bandwidth and buffer capacity is increased by 3 Mbps and 80 packets, respectively. The propagation delay of the bottleneck link is fixed at 10 milliseconds. In these experiments, the multicast sessions starts at uniformly distributed random times within the first 5 seconds of the simulation.

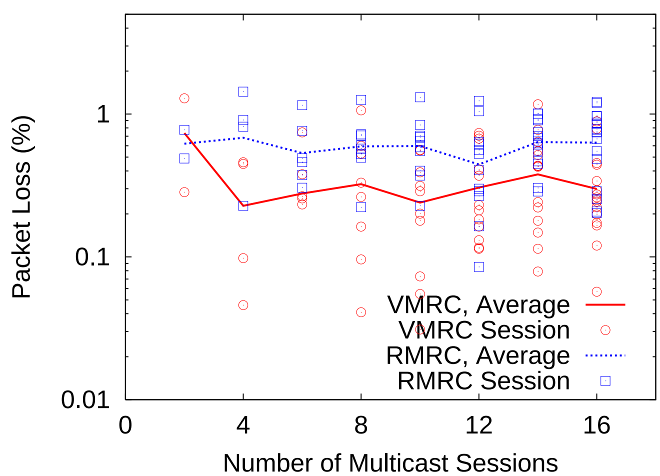
Figure 4.5 shows the average throughput and steady state packet loss rate of each of a number of competing multicast sessions, along with the mean throughput and loss rate over all the sessions. Typically, a fair division of bandwidth is achieved between the competing sessions, with the average throughput of a session being within a factor of two of the average throughput of the other competing sessions. Unlike experiments in the previous section, the VMRC sessions do experience some packet loss, mainly due to the bursty data transmission policy. However, on average, the VMRC sessions experience about a factor of two fewer packet losses than the RMRC sessions.

4.3.4.3 UDP Background Traffic

The results presented in the previous sections demonstrate that the VMRC protocol quickly converges clients behind a common bottleneck link to the sustainable subscription level and operates without inducing packet losses when there is no other flow sharing the bottleneck link. This section evaluates how VMRC performs when it competes with an uncontrolled UDP traffic.



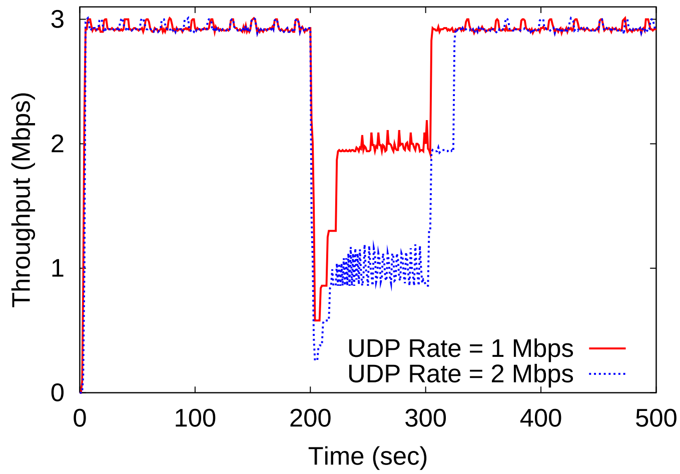
(a) Throughput Share



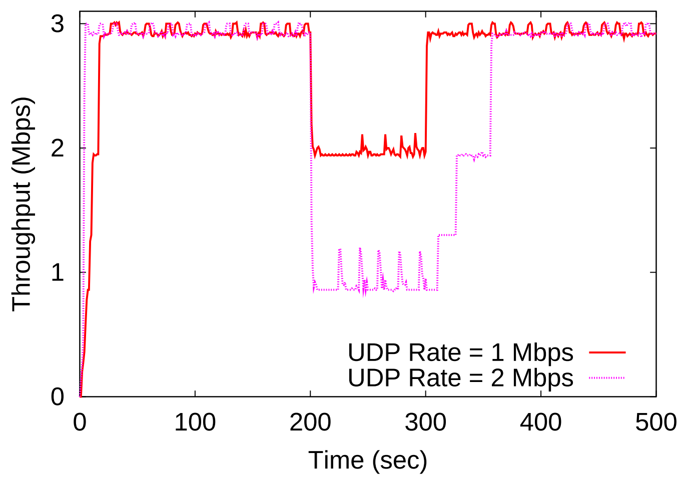
(b) Packet Loss

Figure 4.5: Fairness among Competing Multicast Sessions (bottleneck link of capacity $1.5n$ with a drop-tail queue of $80n$ shared by n multicast sessions)

To evaluate the reaction of VMRC to simple changes in background traffic intensity, experiments were conducted on a 3 Mbps bottleneck link with a 80 packet buffer that is shared between a multicast session and a UDP flow. The multicast session starts at time 0, and two clients join the session at times uniformly distributed between 0 and 2 seconds. The round-trip propagation delays of the two clients are 50 and 150 milliseconds, respectively. The competing UDP flow begins 200 seconds after the multicast session begins, and stops at time 300 seconds. The round-trip propagation delay for the UDP flow is 100 milliseconds. Figure 4.6 shows



(a) VMRC



(b) RMRC

Figure 4.6: Single Multicast Session Sharing Link with a Single Uncontrolled UDP Flow (bottleneck link of capacity 3 Mbps with a drop-tail queue of 80 packets)

the throughput obtained by the multicast sessions as a function of time.

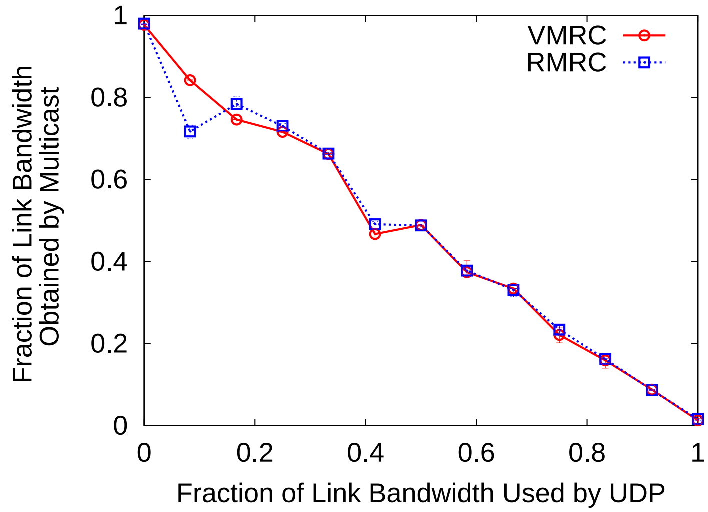
The results show the clients can quickly respond to sudden increases in congestion and drop their throughput share. The VMRC session appears to be more sensitive to changes in network conditions. For example, when a UDP flow of rate 2 Mbps is introduced, the share of the VMRC session drops to 500 Kbps, while that of the RMRC session drops to 1 Mbps. The VMRC session is more responsive to sudden changes in the network conditions as it reacts to both packet losses and increased queuing, while the RMRC session only responds to packet losses. After the

2 Mbps UDP flow stops, the RMRC session takes 60 seconds to utilize the available bandwidth, while the VMRC session ramps up its bandwidth in 30 seconds. The VMRC session is more aggressive in seeking available bandwidth as a small decrease in the queuing delay along the transmission path can translate into a substantial increase in the computed throughput share. In contrast to VMRC, the RMRC session needs to receive enough data packets after the UDP flow stops such that the ALI loss event rate measure is low enough to result in a computed throughput share that enables joining additional layers.

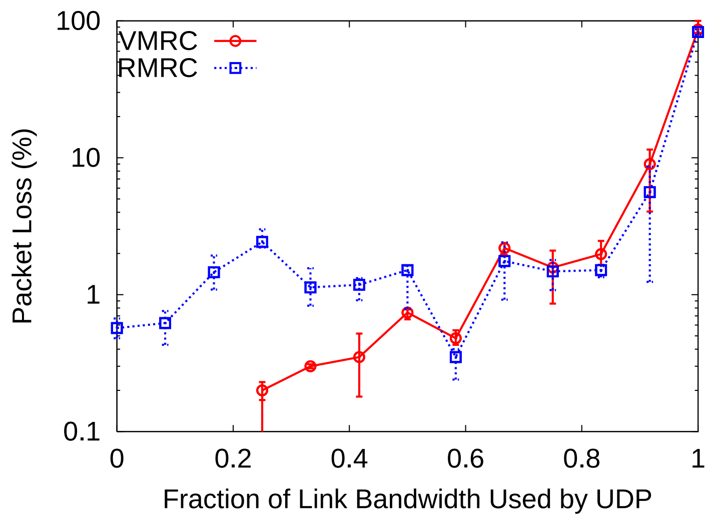
The next experiment considers the impact of varying the rate of the competing UDP flow on the multicast session's performance. Both the multicast session and the UDP flow start at uniformly distributed random times between 0 and 5 seconds, and share a bottleneck link of capacity 3 Mbps with a buffer of 80 packets. The multicast session has 16 clients with round-trip propagation delays uniformly distributed between 40 and 100 milliseconds. The clients join the multicast session at random times that is within 2 seconds of the start of the multicast session. The round-trip propagation delay of the UDP flow is 50 milliseconds. Figure 4.7(a) shows the fraction of the link bandwidth obtained by a multicast session as a function of the fraction of link bandwidth used by the UDP flow. As expected, both RMRC and VMRC sessions obtain the amount of link bandwidth left unused by the UDP flow.

Figure 4.7(b) graphs the steady state loss rates for the multicast session as a function of the link bandwidth used by the UDP flow[1]. The steady state loss rate experienced by a VMRC session is less than that of experienced by a RMRC session, when the competing multicast session is using a significant fraction (e.g., $\geq 50\%$) of the link bandwidth. Also, note that if the fraction of link bandwidth used by the UDP flow is less than 30%, the VMRC session operates without inducing packet losses. When the VMRC session is utilizing a large fraction of the link bandwidth, it can quickly detect any congestion it is introducing in the network because of its

[1]The lines are not smooth owing to the coarse granularity of coarse-grained rate structure of the layering scheme.



(a) Throughput Share



(b) Packet Loss

Figure 4.7: Single Multicast Session Sharing Link with Uncontrolled UDP Flow of Varying Rates (bottleneck link of capacity 3 Mbps with a drop-tail queue of 80 packets)

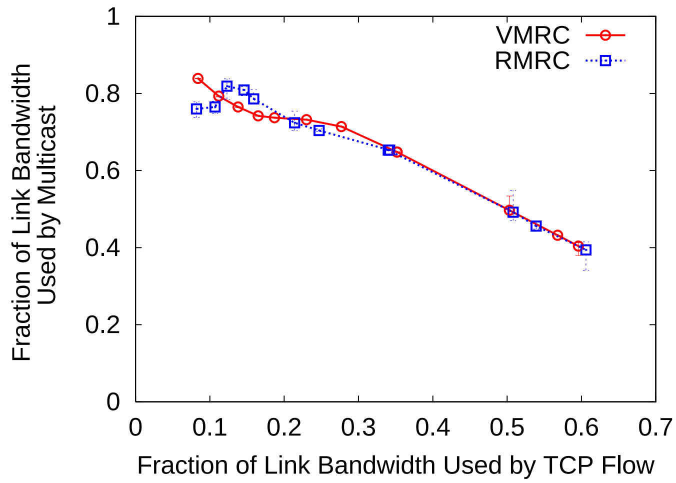
join experiments. This result demonstrates that a rate control protocol based on the TCP Vegas model has the ability to operate without causing significant congestion in the network.

4.3.4.4 TCP Background Traffic

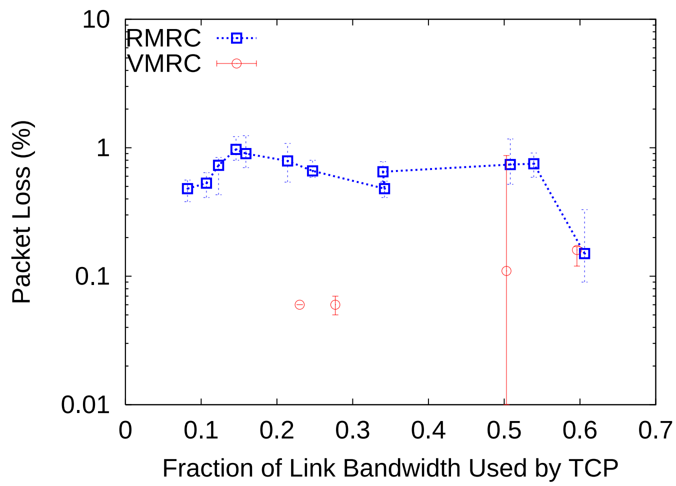
The previous section demonstrated that VMRC sessions are able to utilize the remaining bandwidth when competing with non-rate controlled flows. This section explores the TCP fairness properties of VMRC when sharing the bottleneck link with long duration FTP and ON/OFF HTTP background traffic.

The first experiment in this section is similar to that in Figure 4.7, but with the UDP flow replaced by a FTP flow. The fraction of the link bandwidth that may, at most, be used by the FTP flow is controlled by changing the maximum congestion window parameter of FTP. The FTP flow has a round-trip propagation delay of 100 milliseconds, and the multicast session has 16 clients, with round-trip propagation delay uniformly distributed between 40 and 100 milliseconds. Figure 4.8 shows the throughput share and packet loss rates of the multicast flow, as a function of the link bandwidth used by the FTP flow. Observe that both multicast sessions obtain a greater share of the bottleneck bandwidth when the FTP flows are rate limited. When the FTP flow is not rate limited, the FTP flow can get at most 60% of the link bandwidth. Thus, both multicast sessions are able to share bandwidth reasonably fairly with TCP. However, unlike the RMRC session, the VMRC session experiences no packet losses in a majority of the simulations as it is able to quickly detect unsustainable layer additions. The VMRC session experiences packet losses in some experiments because of synchronization of the TCP and the multicast sources.

So far, the experiments have considered a low multiplexing environment, and the results indicate that VMRC sessions can potentially operate without inducing packet losses. For the remainder of the experiments in this section, the performance of multicast sessions is assessed under a rich mix of background traffic. Background traffic consists of long duration FTP sessions and/or ON/OFF HTTP sessions. Unless stated otherwise, the round-trip propagation delay of the background flows is uniformly distributed between 20 and 460 milliseconds. The foreground flow can be either a long duration FTP flow with round-trip propagation delay of 80 ms or a multicast session with 16 clients with round-trip propagation delays to the server



(a) Throughput Share

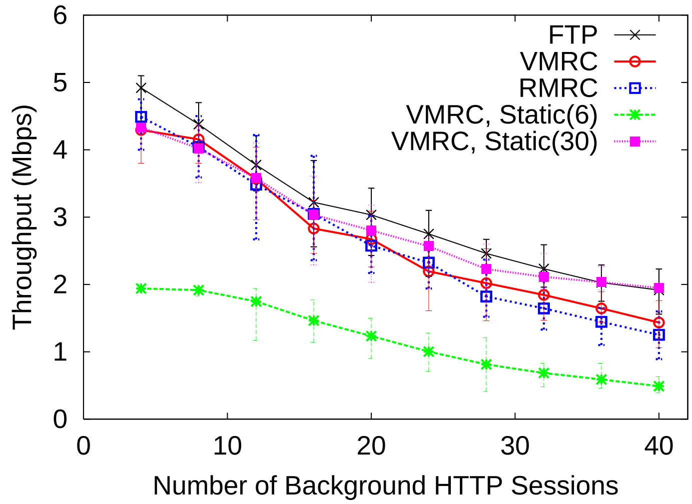


(b) Packet Loss

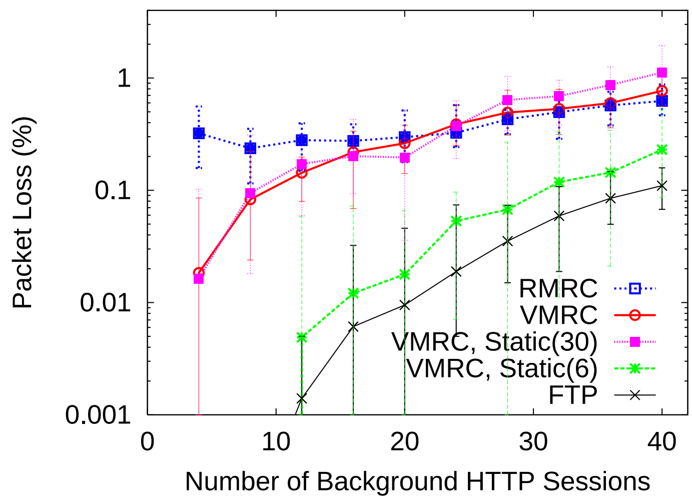
Figure 4.8: Single Multicast Session Sharing Link with a Single FTP Flow (bottleneck link of capacity 3 Mbps with a drop-tail queue of capacity 80 packets)

uniformly distributed between 20 and 460 milliseconds. For a selected background traffic mix, the simulation is run 10 times with the foreground flow of interest, and the average, minimum, and maximum of the metric of interest is graphed.

Figure 4.9 explores how a multicast session performs when competing with HTTP background traffic on a slightly higher bandwidth link of capacity 6 Mbps with a queue that can buffer 150 packets. When the bottleneck link is lightly loaded (less than 20 HTTP flows in this simulation), note that although both RMRC and VMRC sessions obtain similar throughputs, the packet loss experienced by the VMRC ses-



(a) Throughput Share



(b) Packet Loss

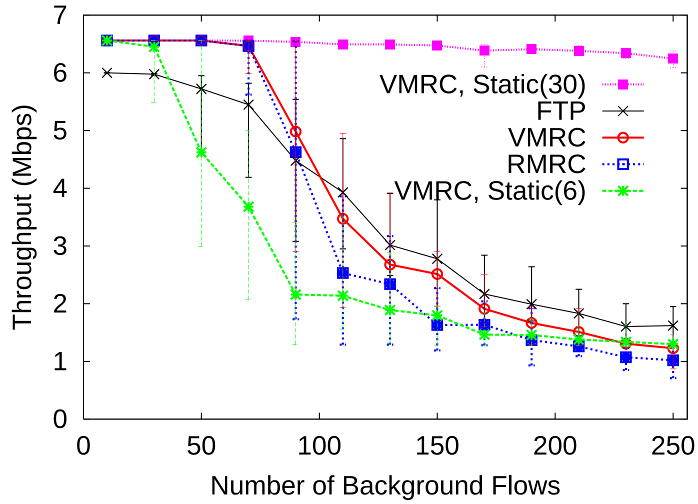
Figure 4.9: Single Multicast Session Sharing a Low Bandwidth Bottleneck Link with Background HTTP Traffic (bottleneck link of capacity 6 Mbps with drop-tail queue of 150 packets)

sion is less than that experienced by the RMRC session. Under these lightly loaded conditions, the background flows can utilize up to 60% of the link bandwidth, and most of the congestion is introduced by the multicast session itself. The results also show that the VMRC session achieves a higher share of the bottleneck link than the RMRC session. Finally, it is worth noting that the acknowledgment clocked nature of TCP transmissions results in much lower packet loss rates than those for multicast sessions, which are transmitting packets in bursts.

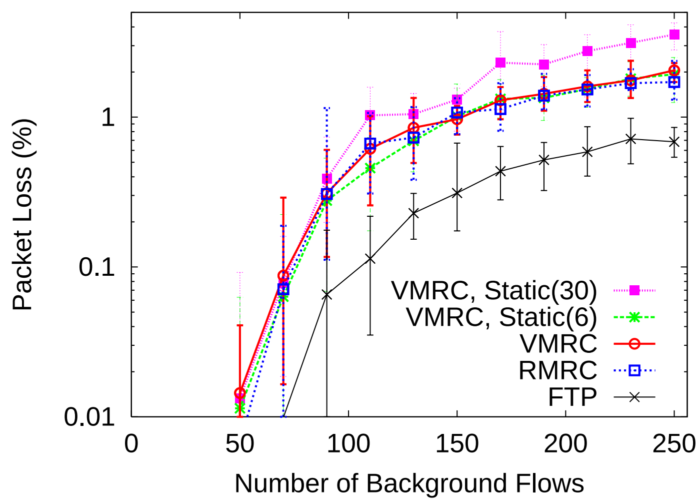
The results also show that the dynamic threshold estimation technique helps maintain TCP-friendliness across a wide range of network loads, as demonstrated by similar throughput shares of the VMRC session and the long duration foreground FTP flow under similar conditions. The results in Figure 4.9 show that a static threshold parameter of 30 can very closely track the throughput obtained by the VMRC session that utilizes dynamic threshold estimation, but with a static threshold of 6, the VMRC session, on average, obtains less than half of the fair share bandwidth. The experiments that follow will further investigate the difficulties in statically setting the Vegas threshold parameters.

Figure 4.10 shows the throughput and packet loss rates of a foreground RMRC session, a VMRC session, and a TCP session, as a function of the total number of background flows for a bottleneck link of capacity 45 Mbps with a queue of 250 packets. The background flows consist of a mix of 90% HTTP and 10% FTP sessions. The results show that for fewer than 50 background flows, there is very little congestion in the network, and the foreground flows obtain the maximum possible throughput (Both the FTP and multicast flows can obtain a maximum throughput of approximately 6.5 Mbps). As the load in the network increases, the throughput share of the foreground flows decreases. Note that for the same settings of multicast protocol parameters, the VMRC session obtains higher throughput than the equivalent RMRC session. The results also show that at high levels of statistical multiplexing, both VMRC and RMRC protocol variants experience similar loss rates, as the packet losses in the bottleneck are due to the background traffic and the bursty transmission policy of the multicast server. As observed before, the FTP flow experiences fewer packet losses than the multicast sessions since FTP transmits packets in response to acknowledgments, and therefore, is more responsive to short-term congestion.

The results further illustrate the difficulty associated with setting the threshold parameters statically. With a static threshold of 30, the VMRC session achieves a throughput share of approximately 6.5 Mbps for the full range of background loads explored in the experiments (i.e., the multicast session obtains a factor of 6 more



(a) Throughput Share

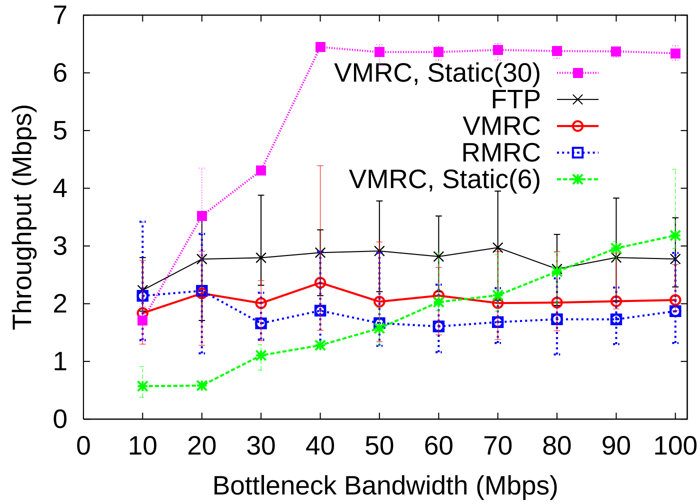


(b) Packet Loss

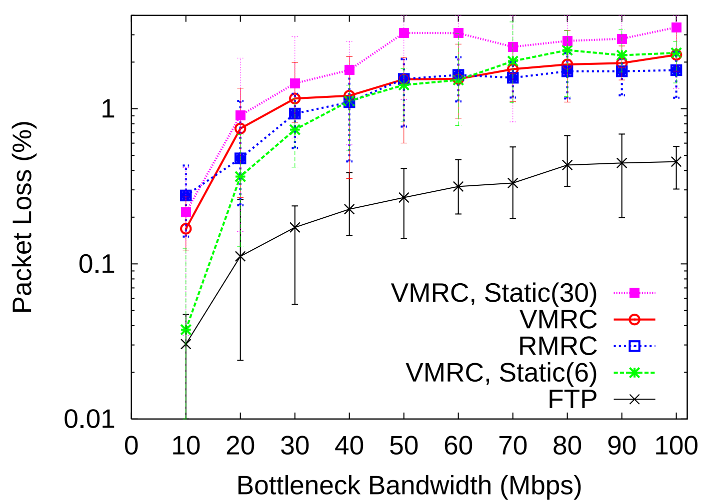
Figure 4.10: Single Multicast Session Sharing a High Bandwidth Bottleneck Link with Background HTTP & FTP Traffic (bottleneck link of capacity 45 Mbps and a drop-tail queue of 250 packets with background traffic consisting of 90% HTTP and 10% FTP sessions)

throughput than FTP). A static threshold of 6 is much less aggressive when the link is lightly loaded (fewer than 70 background flows in this experiment).

Figure 4.11 shows the average throughput and loss rates of the foreground flows for a range of bottleneck link resources. The initial configuration considered is a 10 Mbps bottleneck link with a buffer of 100 packets and with 30 background flows traversing the bottleneck link. The background flows consists of 10% long duration



(a) Throughput Share



(b) Packet Loss

Figure 4.11: Effect of Scaling Bottleneck Link Resources (Background traffic consists of a mix of 90% HTTP and 10% FTP sessions)

FTP sessions and 90% ON/OFF HTTP sessions. For every new experiment, the bottleneck link capacity is increased by 10 Mbps, buffer capacity is increased by 100 packets, and the number of background flows is increasedby 30 (keeping the same mix of background traffic). The results show that the throughput share of multicast and FTP sessions is approximately constant, with the VMRC session obtaining slightly more throughput than the RMRC session. Furthermore, the results illustrate that the dynamic threshold estimation technique works reasonably well over a wide range of bottleneck bandwidth capacities and background traffic mix.

4.3.5 Protocol Design Factors

This section evaluates the performance of the VMRC protocol along the following four protocol design dimensions:

- Synchronization policy: Two alternatives are evaluated: 1) weak synchronization in which join experiments (to any layer) are performed only on receipt of a sender positioned marker in the data stream; and 2) an unsynchronized policy in which clients space their join experiments according to their own local timings.

- RTT estimation policy: Three alternative techniques for RTT estimation are considered: 1) a fixed value for all clients (chosen to be 100 milliseconds in the experiments whose results are reported); 2) a variable estimate that is the sum of the transmission path queuing delay and a fixed value (the fixed value is chosen as 100 milliseconds); and 3) a variable estimate that is the sum of the one-way packet delivery delay plus an estimate of the return path propagation delay. These three techniques are referred to as RTT Fixed, RTT Fixed + Queuing, and RTT Measured, respectively, in the description of the results.

- Data Transmission Policy: The impact of the data transmission policy is studied by varying the interval δ at which packet transmissions are scheduled. Three values for δ are considered: 25, 50, and 100 milliseconds.

- Protocol Reactivity: The influence of reaction speed on protocol performance is studied by varying the slot duration τ. Three values for τ are considered: 100, 200, and 400 milliseconds.

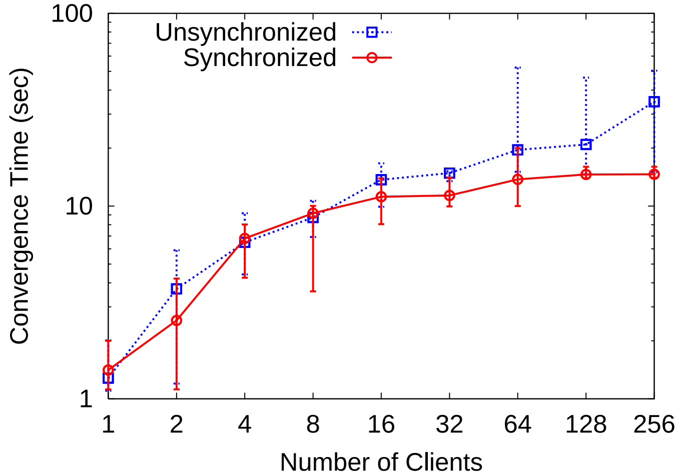
To keep the number of experiments manageable, a simple experimental design is used. That is, to determine how a particular factor affects performance, the experiments consider several choices for that factor while using the default configuration for the remaining three factors.

Two specific simulation setups are repeatedly considered in this section. To study how a particular design affects VMRC session scalability, the simulation setup assumes a 3 Mbps bottleneck link with a 10 millisecond propagation delay and a drop-tail queue of 80 packets. A single multicast session with a number of clients with round-trip propagation delays to the server distributed between 40 and 100 milliseconds and no competing background traffic is assumed. The performance of the VMRC protocol with HTTP background traffic is studied using a bottleneck link of capacity 10 Mbps with a propagation delay of 5 milliseconds and a drop-tail queue of 150 packets. The VMRC session has 16 clients with the round-trip propagation delay of the multicast clients and the HTTP clients uniformly distributed between 20 and 460 milliseconds. The graphs show the average, minimum, and maximum values of the performance metrics obtained from 10 simulation runs.

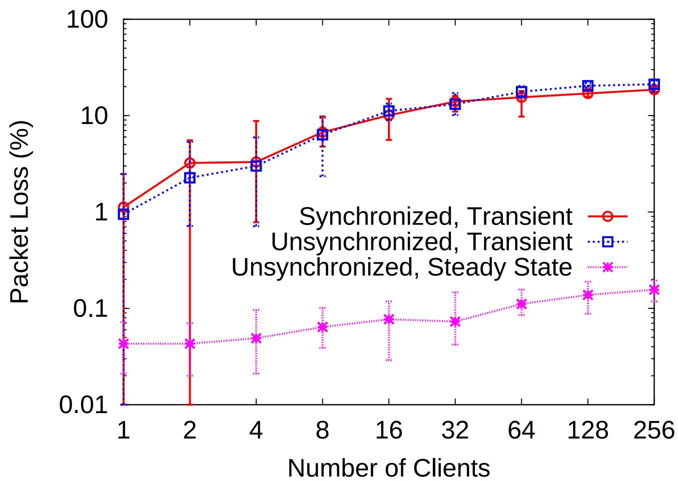
4.3.5.1 Synchronization Policy

Figure 4.12 shows the impact of the synchronization policy on the scalability of a VMRC session. For small sessions (e.g., fewer than 8 clients), the convergence times of the synchronized and the unsynchronized protocols are similar. For larger sessions, the convergence time with unsynchronized protocol is longer than with the synchronized protocol. Note that the synchronized protocol operates without inducing packet losses in the steady state, while the steady state packet loss rate of the unsynchronized protocol increases with increase in the number of clients in the session. With the unsynchronized protocol, clients may join an unsustainable layer at close together but slightly different times, which in turn increases the duration of time the session is subscribed to the unsustainable layer. As the number of clients in the session increases, the possibility of being subscribed to the unsustainable layer also increases, resulting in more frequent buffer overflows. These longer periods of congestion experienced by the unsynchronized protocol increases the convergence time.

Figure 4.13 shows how the synchronization policy affects protocol performance when sharing a bottleneck link with background HTTP sessions. The results show



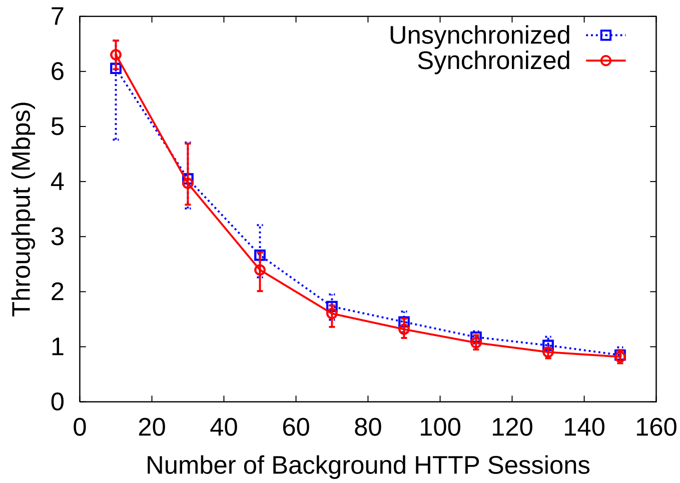
(a) Convergence Time



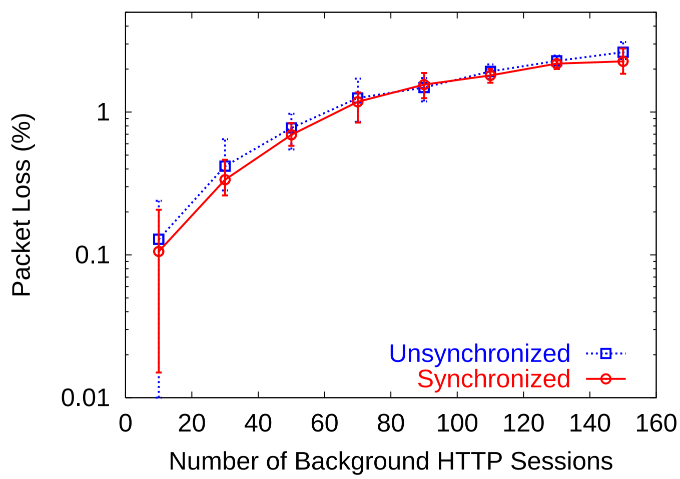
(b) Packet Loss

Figure 4.12: Impact of the Synchronization Policy on the Scalability of a VMRC Session (bottleneck link of capacity 3 Mbps with a drop-tail queue of 80 packets)

that the throughput share and loss rates of the two multicast protocol variants are very similar, with the unsynchronized protocol performing slightly worse than the synchronized protocol when the number of background flows is small. As might be expected, the results from these two experiments demonstrate that the synchronization policy is of importance in low multiplexing environments where the traffic from the multicast session can dominate the bottleneck queue dynamics.



(a) Throughput Share



(b) Packet Loss

Figure 4.13: Impact of the Synchronization Policy on VMRC Performance when Sharing Bottleneck with Background HTTP Sessions (bottleneck link of capacity 10 Mbps and a drop-tail queue of 150 packets)

4.3.5.2 RTT Estimation Policy

Consider a VMRC session with three clients sharing a bottleneck link of capacity 10 Mbps with a varying number of HTTP background sessions. The three clients have round-trip propagation delays of 20, 200, and 400 milliseconds to the source. Figure 4.14 shows the throughput share of the clients when the multicast protocol is actively tracking the RTT. The results illustrate that actively tracking RTT may be undesirable as clients behind a common bottleneck link but with markedly different

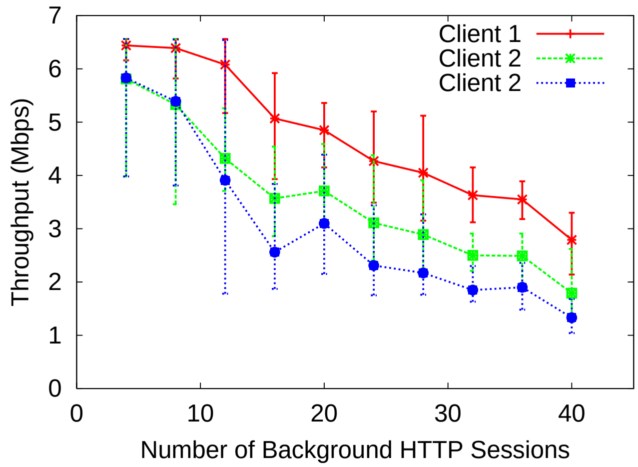
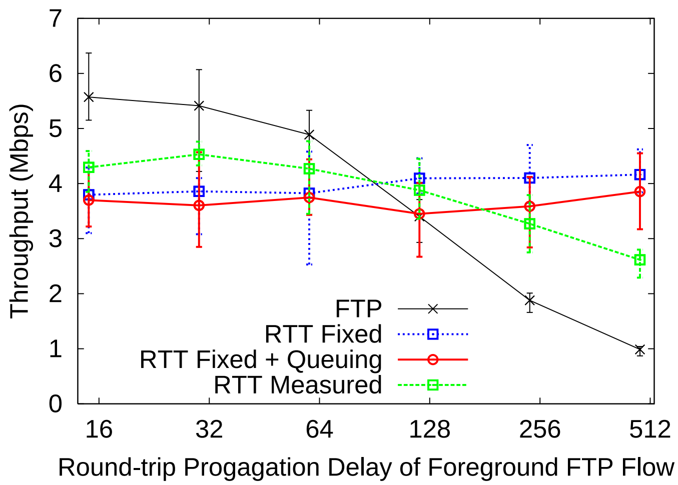


Figure 4.14: Throughput Share of Three Different Clients Behind the Same Bottleneck Link when RTT is Explicitly Tracked (bottleneck link of capacity 10 Mbps with a drop-tail queue of 150 packets)

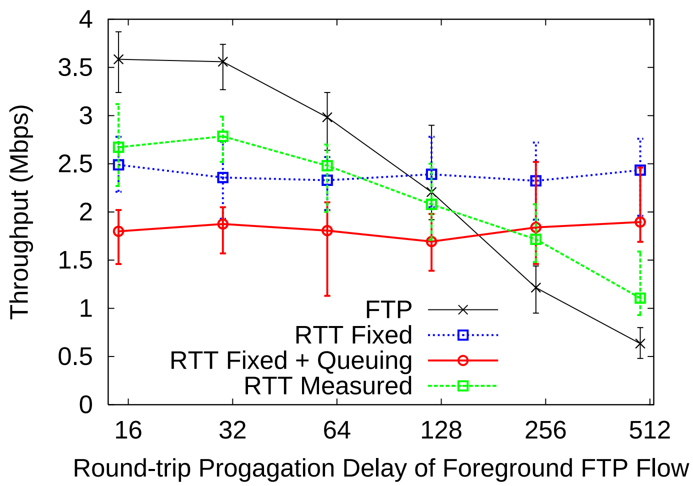
round-trip propagation delay, such as in this experiment, do not converge to a common subscription level.

The next experiment explores the TCP fairness properties with the three RTT estimation techniques. The simulation uses a 10 Mbps bottleneck link with a drop-tail buffer of 150 packets. The foreground flow can be a single client VMRC session or a FTP flow utilizing Newreno TCP. Figure 4.15 graphs the throughput share of the foreground flow for a range of selected foreground round-trip propagation delays.

The results show that the throughput achieved with TCP Newreno has a strong dependence on the round-trip propagation delay. Although the throughput achieved with a VMRC session that actively tracks RTT also exhibits some dependence on round-trip propagation delay, the dependence is not as pronounced as that with TCP Newreno. For example, observe from Figure 4.15(a) that the throughput of the TCP Newreno varies between 1 and 6.5 Mbps, while that of a VMRC session that actively measures RTT varies between 2.2 and 4.6 Mbps. This somewhat lesser dependency on the RTT suggests that estimating RTT as a fixed value or as the sum of a fixed value and a variable queuing delay term might achieve reasonably good TCP fairness. For the configurations considered in this experiment, a VMRC session that estimates RTT as the sum of a fixed component and a variable queuing delay



(a) 30 HTTP Sessions

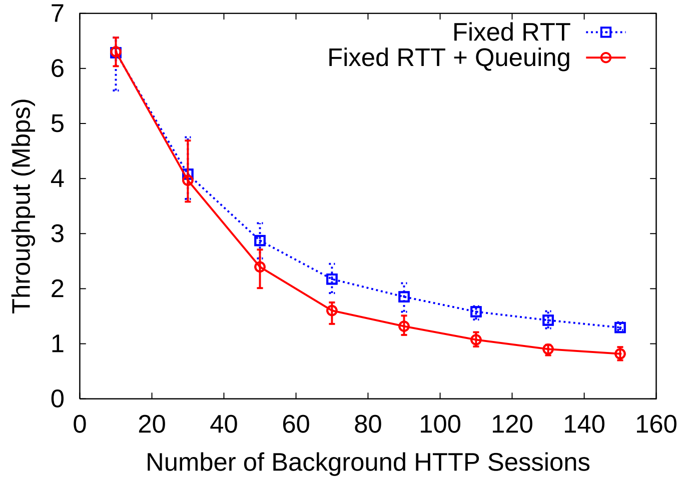


(b) 60 HTTP Sessions

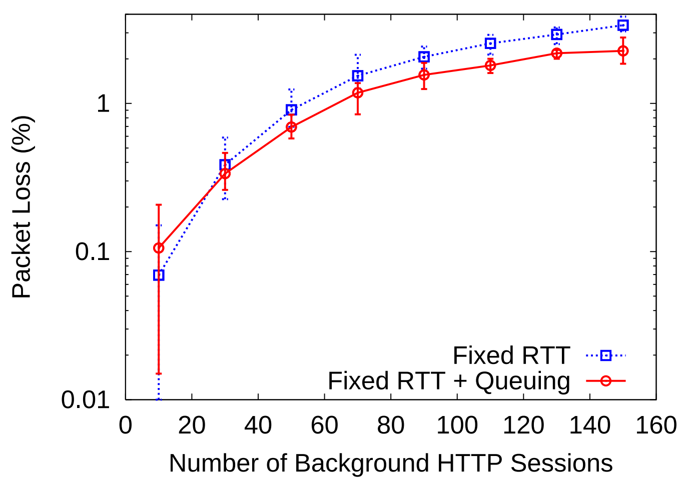
Figure 4.15: Illustrating the TCP Fairness Properties of VMRC Sessions (bottleneck link of capacity 10 Mbps with a drop-tail queue of 150 packets)

component can achieve throughput that is within a factor of two of that obtained by tracking the actual RTT.

Figure 4.16 shows how the throughput share obtained by a VMRC session is affected by the RTT estimation policy when competing with background ON/OFF HTTP traffic. With an increase in the number of background HTTP sessions, the bottleneck queuing increases and the fixed RTT approach becomes relatively more aggressive.



(a) Throughput Share

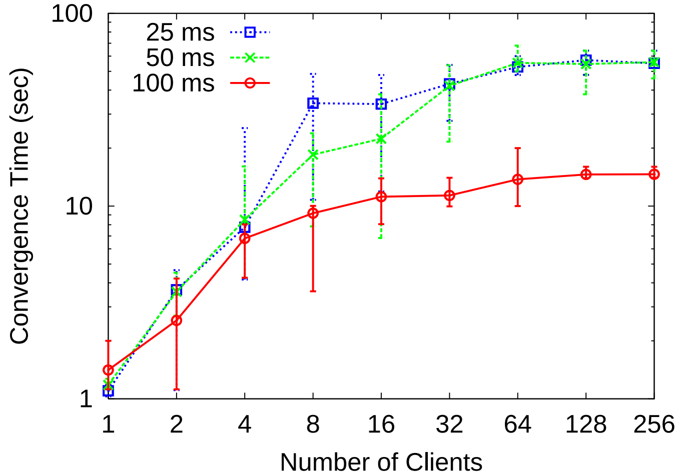


(b) Packet Loss

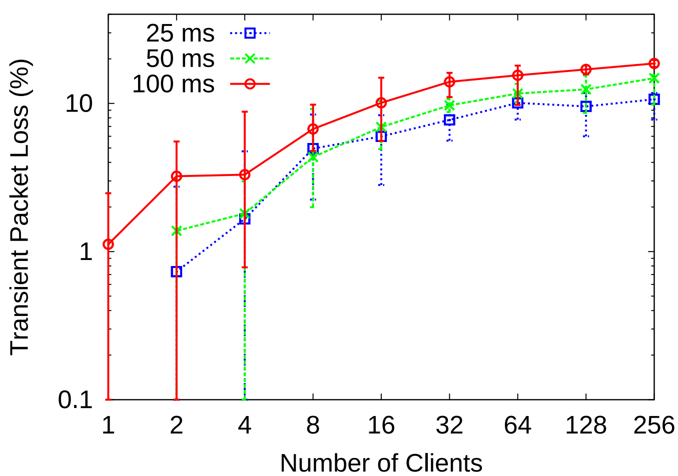
Figure 4.16: Impact of the RTT Estimation Policy on VMRC Performance when Sharing Bottleneck with HTTP Background Flows (bottleneck link of capacity 10 Mbps with a drop-tail queue of 150 packets)

4.3.5.3 Transmission Scheme

Figure 4.17 shows the influence of the data transmission burst size on convergence times and transient packet loss rates. The results show that although a smaller burst duration of 25 milliseconds results in a factor of three to five times less transient loss than with a burst duration of 100 milliseconds, there is some penalty involved in terms of increased convergence times. Also observe the that reduction in transient packet losses is much more substantial when the burst duration is reduced from



(a) Convergence Time

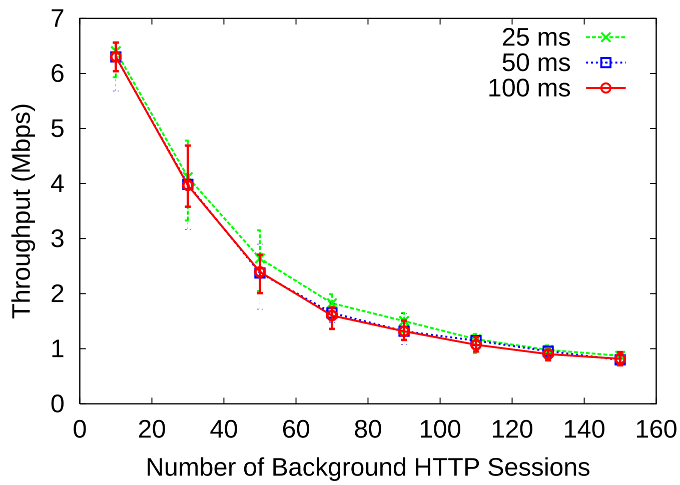


(b) Packet Loss

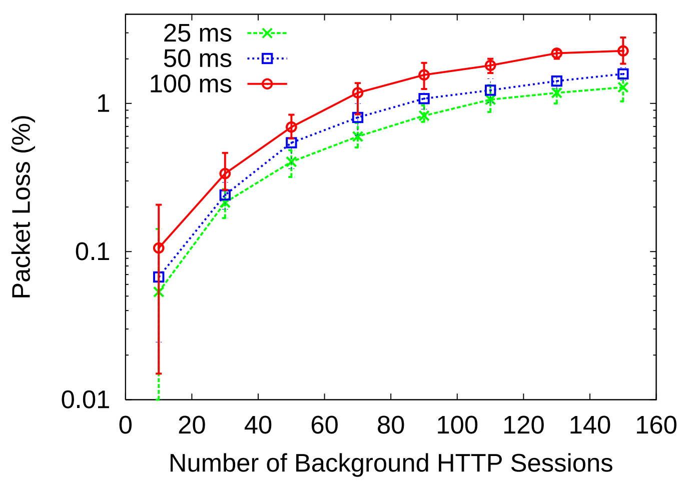
Figure 4.17: Impact of the Data Transmission Policy on VMRC Session Scalability (bottleneck link of capacity 3 Mbps with a drop-tail queue of 80 packets)

100 milliseconds to 50 milliseconds, then when it is reduced from 50 milliseconds to 25 milliseconds. This behavior, along with the system overhead associated with increased scheduling frequency, should be considered when implementing the protocol for production use.

Figure 4.18 evaluates the alternatives with background HTTP traffic. It is interesting to note that in this scenario, the transmission burst size has only a little impact on the throughput share. However, smaller bursts can somewhat reduce the packet loss experienced by the multicast session.



(a) Throughput Share



(b) Packet Loss

Figure 4.18: Impact of the Data Transmission Policy on VMRC Performance when Sharing Bottleneck with Background HTTP Sessions (bottleneck link of capacity 10 Mbps with a drop-tail queue of 150 packets)

4.3.5.4 Reactivity

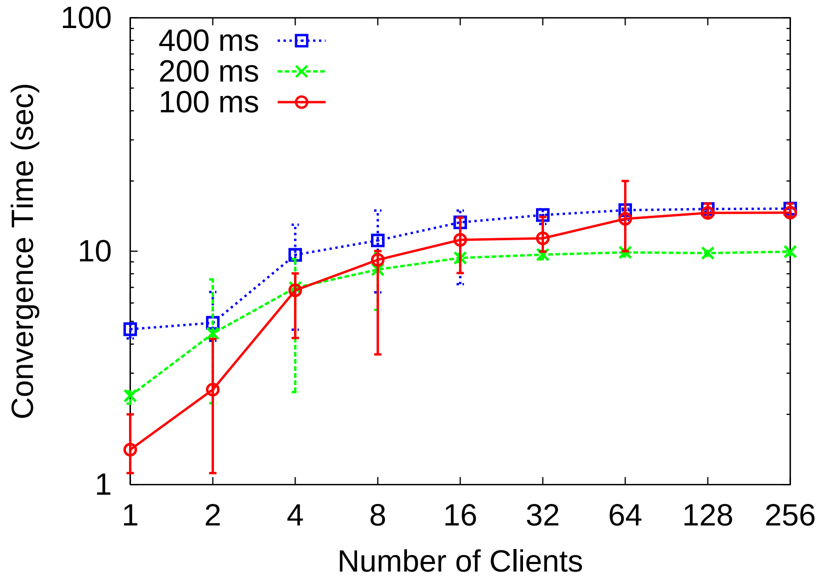
The final design factor considered is the reactivity of the protocol. Note that although several parameters such as the weights used in the ALI and EWMA techniques, time slot duration, and placement of add markers affect protocol performance, reasonable insight concerning the impact of reactivity can be obtained by simply changing the duration of the time slot. Note that all long-term measurements occur at time slot boundaries, and, therefore, increasing/decreasing time slot duration impacts how slowly/quickly the throughput equation input parameters change. It should be noted, however, that the short-term measures are still being updated on a per-packet basis.

Figure 4.19 illustrates the impact of reactivity on convergence and packet loss. For the time slot durations considered, the protocol reactivity appears to have only nominal impact on convergence time. However, the slower the protocol is in reacting to congestion, the more packet losses it will experience. Note that the plots for the transient packet losses need to be interpreted with care, since transient loss is computed over the start-up duration of clients. As the time slot duration is scaled, the startup duration is also appropriately scaled up (i.e., with $\tau = 100, 200, 400$ milliseconds, the startup durations are $5, 10, 20$ seconds, respectively).

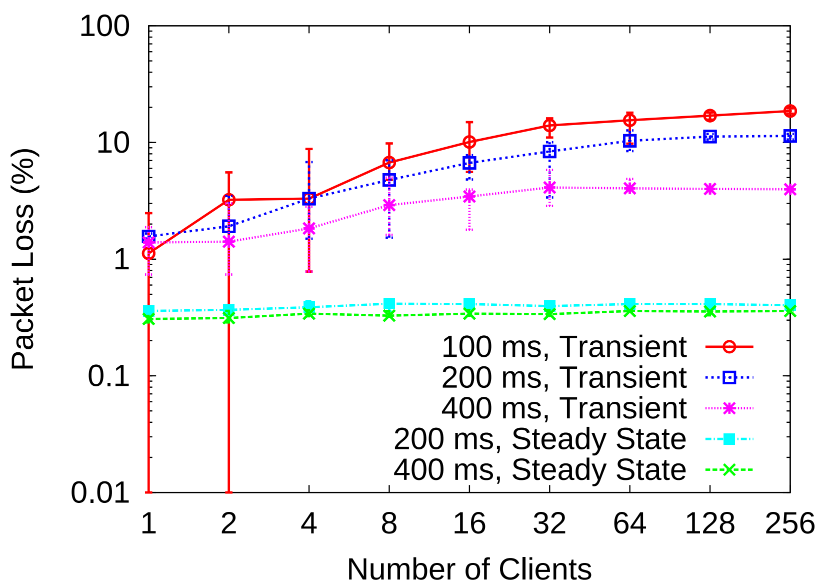
For large numbers of clients, the transient packet loss rate with $\tau = 100$ milliseconds is approximately 10%, and with $\tau = 400$ milliseconds the transient loss rate is approximately 4%. Since the multicast session receives four times as much more data when $\tau = 400$ milliseconds, the actual number of packets lost in this case is approximately a factor of 1.6 times more than that with $\tau = 100$ milliseconds. Also, as expected with longer time slots, the multicast session is not able to operate in "loss free" mode.

Figure 4.20 considers the impact of protocol reactivity when sharing a bottleneck link with background HTTP sessions. It is interesting to note that decreasing the reactivity of the protocol did not significantly degrade performance. One possible explanation is that with a less reactive protocol, the multicast session makes fewer attempts to join additional layers, but at the same time, can stay subscribed to a already subscribed layers for longer durations as it cannot drop the layers as often. It appears that the inability to add layers frequently is adequately compensated by the ability to retain a layer for longer time periods.

The above results illustrate, although indirectly, the benefit of tracking both short-term and long-term estimates of the parameters of the throughput model. For example, when the slot duration of the protocol is long, the clients can still detect an unsustainable layer subscription quickly as the short-term measures are updated on

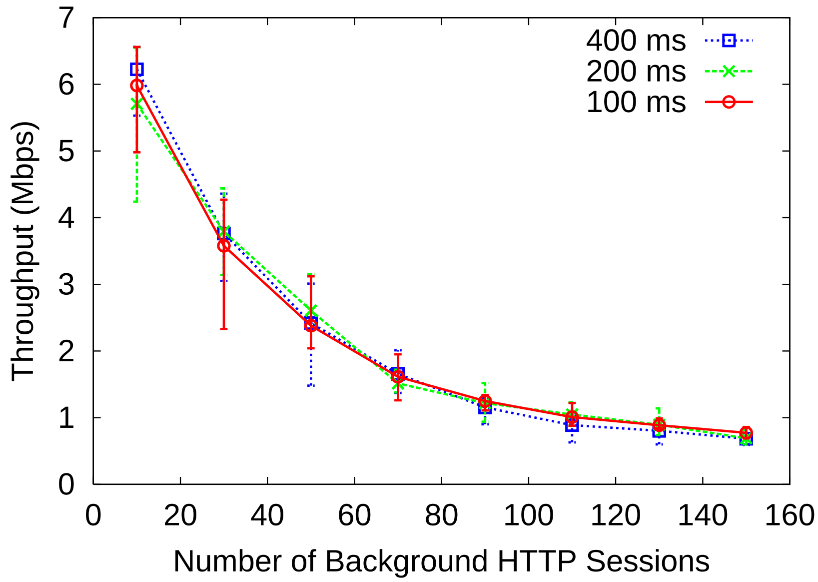


(a) Convergence Time

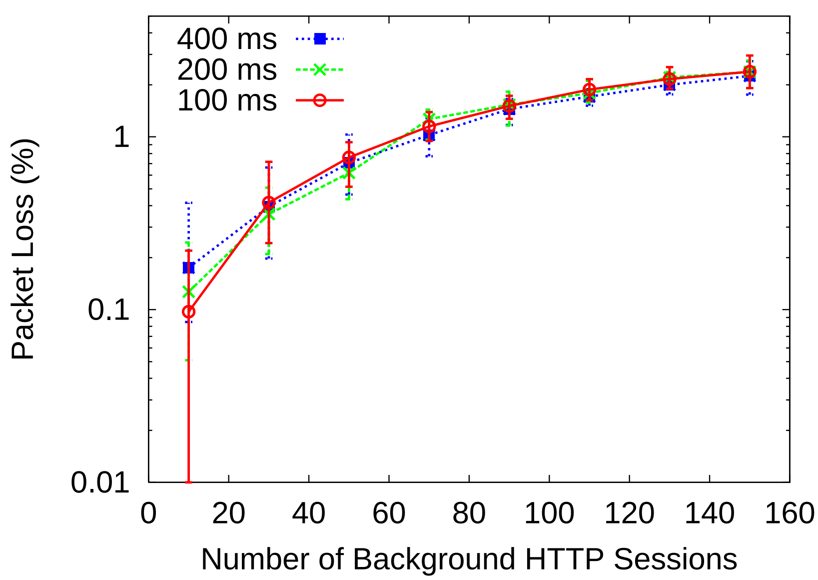


(b) Packet Loss

Figure 4.19: Impact of Protocol Reactivity on the Scalability of a VMRC Session (bottleneck link of capacity 3 Mbps with a drop-tail queue of 80 packets)



(a) Throughput Share



(b) Packet Loss

Figure 4.20: Impact of Protocol Reactivity on VMRC Performance when Sharing Bottleneck with Background HTTP Sessions (bottleneck link of capacity 10 Mbps with a drop-tail queue of 150 packets)

a per packet basis. The drawback of a longer time slot, as well as of slowly updating the long-term estimates, is therefore restricted to slower reactivity to changes in the congestion levels in the network.

4.4 Summary

This chapter developed a new equation-based multirate congestion control protocol, Vegas Multicast Rate Control (VMRC), that uses a recently proposed TCP Vegas throughput model. The VMRC protocol exhibits TCP Vegas-like behavior and operates without inducing packet losses when the bottleneck link is lightly loaded.

The VMRC protocol incorporates a new technique for dynamically adjusting the Vegas threshold parameters based on measured characteristics of the network. This technique implements fair sharing of network resources with other types of competing flows, including widely deployed versions of TCP such as TCP Reno, which is not possible with the previously defined static Vegas threshold parameters.

An extensive performance evaluation study of VMRC is presented, and in order to evaluate the benefits of Vegas-like congestion control, the performance of VMRC is also compared to that of an analogous protocol that is based on a TCP Reno throughput model. A key aspect of the performance study is the evaluation of the design choices made in the VMRC protocol along each of the primary dimensions, namely synchronization policy, RTT measurement, data transmission policy, and protocol reactivity.

Chapter 5

Quality Adaptation

Quality adaptation is the process of altering the media stream received by a client in response to bandwidth available on the path from the server, so that the overall perceived quality is as high as that permitted by the average available bandwidth. The quality adaptation techniques identified in this chapter are classified as *static* or *dynamic*. Dynamic quality adaptation attempts to maximize the playback quality at the client without causing frequent changes in the media quality, given time-varying available bandwidth. Static quality adaptation aims at providing clients with efficient tradeoffs between start-up delay and media quality.

Quality adaptation is particularly difficult when the available bandwidth is time-varying. A naive dynamic quality adaptation technique for such a context is to vary the playback quality directly in response to changes in the available bandwidth. This may result in highly variable playback quality, which is undesirable for users. Alternatively, *work-ahead* buffering of data, that is receiving and buffering data prior to when it is needed for playback, can be used to sustain a more uniform quality of playback. One such scheme has been developed for unicast streaming of layered media files [104]. In this scheme, the server adds or drops media layers in response to long-term changes in the available bandwidth, while work-ahead is used to smooth short-term variations in the available bandwidth. This work-ahead is achieved by transmitting the data for certain layers (determined by the work-ahead algorithm) at a rate higher than its consumption rate during periods of high bandwidth. During periods of low bandwidth, the server can reduce the sending rate on layers that have sufficient work-ahead. Fundamental to the success of this mechanism is the manner in which work-ahead is allocated among the layers. Maintaining the best possible subscription level entails achieving work-ahead on various layers such that a high subscription level can be maintained in the event of short-term changes in available bandwidth. Allocating more work-ahead to lower layers reduces the risk of wasting work-ahead in the event of layer drops, while distributing work-ahead among all the layers allows the maximum short-term reception rate reductions.

The techniques described in [104] for dynamic quality adaptation using work-ahead are not applicable in the context of scalable on-demand multicast streaming protocols. Changing the transmission rate of individual layers is *not* feasible in this context, since transmissions are multicast, and changing the transmission rate on a layer to accommodate the quality adaptation needs of one client, may adversely impact other clients. Also, although it is possible in the context of scalable periodic broadcast protocols such as Skyscraper [62] and Optimized PB [84] to perform work-ahead by starting to listen to a channel earlier than the protocol requires, the available bandwidth may decrease and the channels may have to be dropped before the associated segment is fully received. In this case, work-ahead may not prove useful, since the received portion of segment may not be the initial portion, and in any case, when the receiver again begins to listen to the channel, the portion of the segment currently being received may substantially overlap with the portion already received. The main contribution of this chapter is an efficient dynamic quality adaptation technique using work-ahead for scalable periodic broadcast protocols.

Instead of, or in addition to, work-ahead based dynamic quality adaptation, scalable streaming can make use of static quality adaptation in which clients with some a priori knowledge of their available bandwidth choose a number of media layers to receive. This choice can be made more efficient if there is a tradeoff available between start-up delay and required client data rate. Of course, for non-layered media, such a property is even more desirable. Previous scalable streaming protocols described in Chapter 2, and the protocols developed in Chapter 3, assume that all clients have the same achievable data rate. The segment size progress is tailored for this data rate, and thus a client with slightly lower data rate may require a relatively large increase in the start-up delay to obtain enough work-ahead so that all the media is received prior to its play point. Other contributions of this chapter are the proposed new optimized Heterogeneous Periodic Broadcasts (HPB) protocols that extend the Optimized PB protocols [84] presented in Chapter 3 to efficiently support heterogeneous client data rates. The HPB protocols support flexible static quality adaptation by providing efficient tradeoffs between start-up delay and required client data rate, by allowing the client to efficiently receive more media layers at the cost of increased start-up delay.

The remainder of this chapter is organized as follows. The new HPB protocols and the associated static quality adaptation mechanisms are described in Section 5.1. Section 5.2 develops an efficient dynamic quality adaptation mechanism using work-ahead applicable to HBP systems as well as other periodic broadcast systems. Quality adaptation issues relevant to scalable bandwidth skimming policies are identified in Section 5.3. The chapter is summarized in Section 5.4.

5.1 Static Quality Adaptation

This section considers the problem of efficiently supporting heterogeneous clients in a periodic broadcast system using a single set of periodic broadcast channels. The discussion is applicable for the delivery of a single monolithic media file or a single layer of a media file. New protocols are developed assuming that each client has a fixed data rate for the entire reception of the media file. The new protocols allow the clients to tradeoff start-up delay to receive a higher quality of streaming media. The techniques developed in Section 5.2 may be used in conjunction with these new (as well as other) protocols to accommodate client reception rates that dynamically vary owing to changes in available bandwidth.

5.1.1 Optimized HPB

Existing periodic broadcast protocols can support heterogeneous clients by adding sufficient additional start-up delay for clients with aggregate transmission rates less than that required by the protocol. The additional start-up delay is used to achieve enough work-ahead such that jitter-free playback can occur. This approach is very inefficient because previous periodic broadcast protocols are designed by assuming a specific client data rate. Figure 5.1 illustrates this inefficiency for the Optimized PB protocol with parameters $K = 40$, $b = 2$, $s = 8$, and $r = 0.25$[1], by plotting the required start-up delay (in units of the media duration) as a function of the average client data rate (in units of the media playback rate). The Optimized PB curve shows that clients with achievable data rates between 1.5 and 2 require start-up delay that is a factor of 50 or higher, than that of clients having a data rate equal to 2, for which the protocol is optimized. Note that the line is a step function since clients can use bandwidth only in increments of the channel transmission rate r.

From the discussion of Optimized PB in Chapter 3, it is known that the start-up delay with the protocol approaches the lower bound as the segment transmission rate becomes very small. For fixed K and s, a "tailored" progression can be designed for a client with achievable data rate b_i by setting the segment transmission rate r to b_i/s. Figure 5.2 shows the factor increase in start-up delay with respect to such a tailored progression as a function of the client data rates. It can be seen that clients with data rates less than 2 require start-up delays that are substantially higher than that of the corresponding tailored progression. Figures 5.1 and 5.2 also illustrate the performance improvements possible with the new protocols.

The new Optimized Heterogeneous Periodic Broadcast (Optimized HPB) protocols are optimized for a set of heterogeneous client data rates, rather than being optimized for a single rate. The design of the Optimized HPB protocols is described next.

First, the range of client data rates that the system will most efficiently support is

[1]The notation for Optimized PB is given in Table 3.2 on page 61.

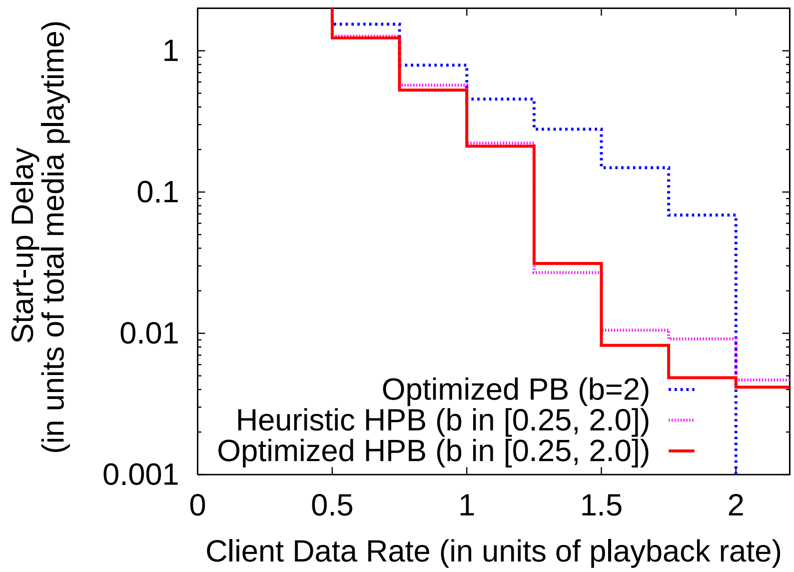


Figure 5.1: Start-up Delay vs. Average Client Data Rate ($K = 40$, $r = 0.25$)

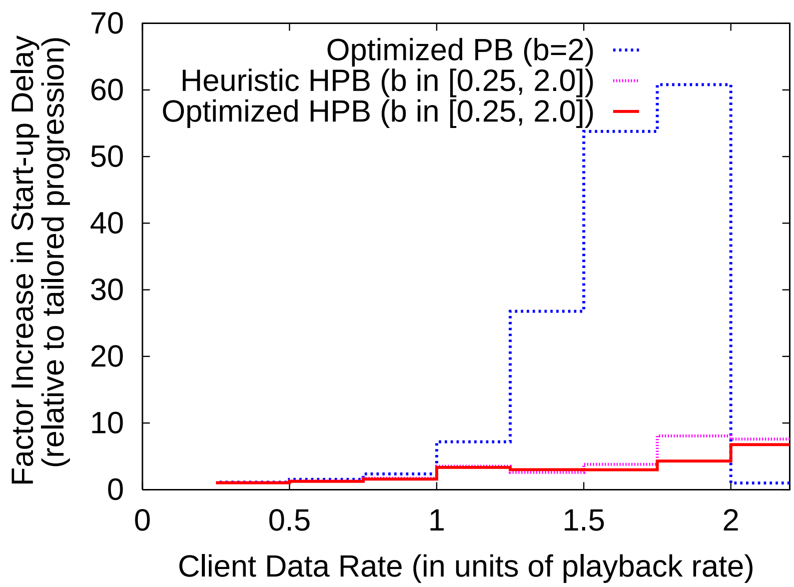


Figure 5.2: Factor Increase in Start-up Delay vs. Average Client Data Rate ($K = 40$, $r = 0.25$)

determined. Heterogeneity in client data rates is supported by designing a segment size progression that minimizes a chosen function of the start-up delay of clients with rates in the selected range. Here, the geometric mean[2] (denoted below by G) is selected as the optimization function. The geometric mean has the property that equal weight is placed on the same factor of improvement in start-up delay, at any of the client data rates of interest, regardless of the magnitude of that start-up delay. However, this approach can accommodate alternative functions, such as a weighted arithmetic mean with weights selected so as to provide specific quality of service to particular classes of clients. Note that G is a type of "average" start-up delay for the range of supported client data rates.

Deriving the exact segment size progression that minimizes G, however, appears to be a hard problem. One approximate approach is to start with the segment size progression for Optimized PB for a client data rate in the range of interest, and modify that segment size progression, using classical techniques (such as simulated annealing, greedy search, etc), until G is minimized. However, it was found that such searches failed as the techniques became trapped in local minima.

An alternative, heuristic approach ("Heuristic HPB") that was found to yield quite good results is as follows. First, the relative segment sizes are computed assuming Optimized PB with the highest client data rate in the selected range. Then, each computed relative segment size is reduced by a particular percentage that is a function of the segment number. Specifically, the last 50% of the segments are reduced in size by some percentage X, the preceding quarter by $X/2$, and the preceding eighth by $X/4$. Using a simple search technique, the value of X that minimizes G can be found. The actual segment sizes as a fraction of the total media file can then be obtained by dividing each relative segment size by the sum of the relative sizes.

A second method ("Optimized HPB") that was found to yield even better results uses the functional form $\alpha k^{\delta} + \beta k^{\delta/2} + \gamma$ to compute the segment size progression,

[2]The geometric mean of d_1, ... d_m is defined as $\sqrt[m]{\prod_{i=1}^{m} d_i}$.

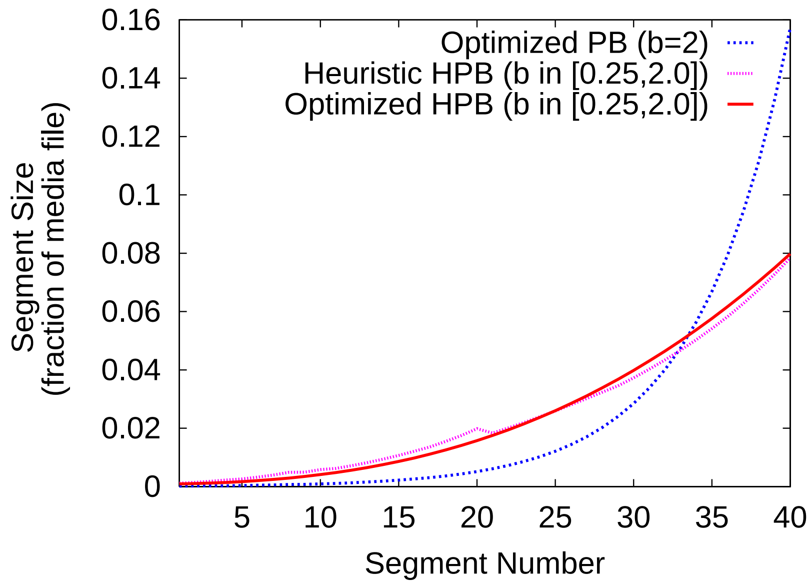


Figure 5.3: Protocol Segment Size Progressions ($K = 40$, $r = 0.25$)

where k is the segment number and α, β, γ and δ are constants that are chosen to minimize G. In most cases, simple search techniques can be employed to determined these constants.

For $K = 40$ and $r = 0.25$, Figure 5.3 shows the optimized segment size progression determined using this method for the range of client data rates in [0.25, 2.0]. Note that although segments are discrete, the figure shows continuous curves for convenience. The functional form used above might appear to contradict the expected exponential growth of segment sizes observed in prior work (e.g., see [59, 84]), and as shown, for example, in Figure 5.3 for the Optimized PB protocol with $b = 2$. However, prior work has considered maximal segment size increases for fixed client data rates, whereas HBP considers a range of client data rates. Accommodating clients with lower data rate obviously requires a slower growth of segment sizes. Segment sizes obtained using the heuristic method also have this polynomial growth of segment sizes. From Figure 5.3, observe that the segment sizes obtained using the heuristic method and the method based on specific functional form are nearly identical.

The start-up delay for the highest client data rate is simply the time to download the first segment. For lower client data rates, the start-up delay is determined by the minimum work-ahead required to ensure that all data is received in time for playback. This can be simply computed in time linear in the number of segments.

Previous periodic broadcast protocols have the desirable property that linear increases in server bandwidth yields exponential decrease in start-up delay, assuming a segment size progression tuned to a homogeneous client data rate. A key question is whether or not this attractive property holds for Optimized HPB systems designed to efficiently support heterogeneous client data rates. Figure 5.4 explores this issue by graphing the required start-up delay for a specific client data rate as a function of server bandwidth, in an Optimized HPB system designed for a client data rate range of [0.25, 2.0] and segment streaming rate $r = 0.25$. As in Chapter 3, server bandwidth is expressed in units of the media playback rate, while start-up delays are expressed as a fraction of the media file playback duration. The results indicate that the above mentioned property largely holds for clients with data rates greater than the media playback rate (i.e., $b > 1$) as well as for the overall geometric mean of the start-up delays. The lines in the figure are not smooth mainly because of the granularity of the search method used to determine the parameters of the polynomial function. For clients with data rates less than the media playback rate, increasing server bandwidth has negligible impact on start-up delay since the client must buffer a large fraction of the media file before playback can begin.

5.1.2 Trading Off Start-up Delay and Quality

Higher quality can be achieved at the client at the cost of increased start-up delay in an Optimized HPB system where each layer of a media file is delivered using a separate instance of the HPB protocol. Each instance of the protocol uses the same parameters K and r (measured in units of the layer bit rate), and has the same segment size progression.

As an example, consider broadcasting a 100 minute video using a 5 layer HPB system with equal bit rate per layer, and parameters $K = 40$ and $r = 0.25$ (implying

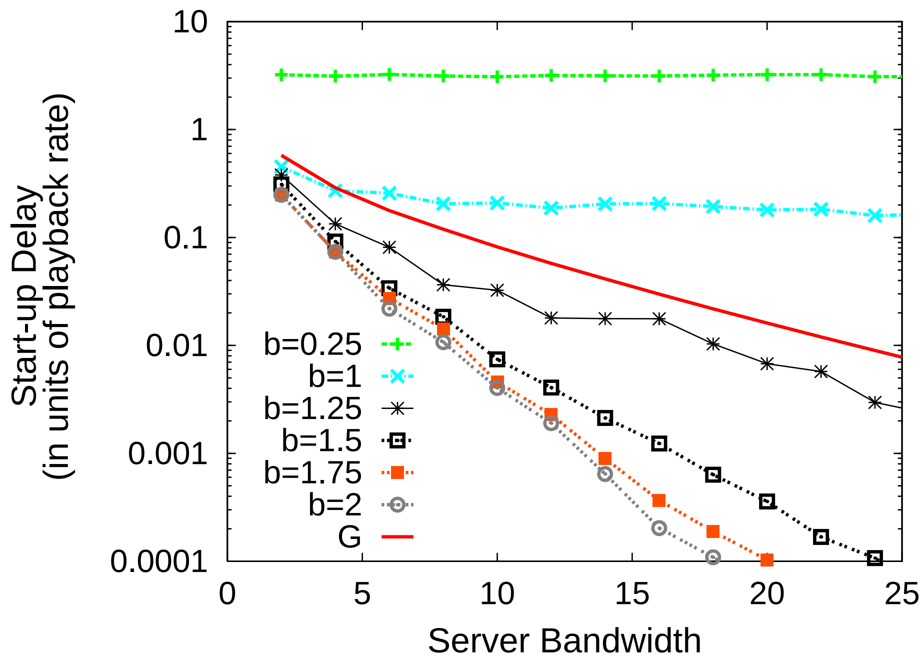


Figure 5.4: Target Delay vs. Server Bandwidth (b in $[0.25, 2.0]$, $r = 0.25$)

a server bandwidth of 10 per layer). Now consider a client with total available bandwidth $b = 5$. Such a client has bandwidth $b = 1, 1.25, 1.66, 2.5$ if only 5, 4, 3, or 2 layers are received, respectively. Thus, from Figure 5.4, it can be seen that the client has a choice of receiving all 5 layers with a start-up delay of 21 minutes, 4 layers with a start-up delay of 3 minutes, 3 layers with a start-up delay of 45 seconds, or 2 layers with a start-up delay of 24 seconds. There is no advantage to receiving just the base layer as it does not reduce the start-up delay beyond 24 seconds. Had the same file been broadcast using the Optimized PB protocol (with $b = 2$), the choices available to the client would be receiving 5 layers with a start-up delay of 45 minutes, 4 layers with a start-up delay of 28 minutes, 3 layers with a start-up delay of 15 minutes, or 2 layers with a start-up delay of 4 seconds. Clearly, Optimized HPB offers a much better range of tradeoffs between media quality and start-up delays.

Figure 5.5 examines more closely the relationship between client data rate and start-up delay with Optimized HPB, for fixed server bandwidth of 10 times the media bit rate and various values of the segment transmission bit rate r. The results show

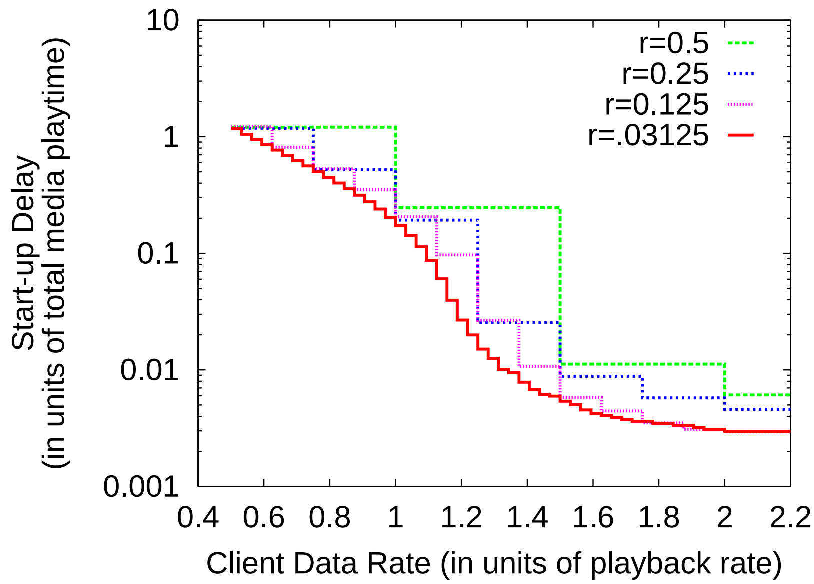


Figure 5.5: Impact of Segment Transmission Rate r ($b = [0.5, 2.0]$, $B = 10$)

that lower segment transmission rates better accommodate heterogeneity in client data rates at the cost of using more multicast channels.

5.2 Dynamic Quality Adaptation

This section describes techniques for quality adaptation using work-ahead when the available bandwidth is time-varying, assuming delivery of layered media. Section 5.2.1 describes an approach for achieving work-ahead in periodic broadcast systems. Policies for performing work-ahead are considered in Section 5.2.2. Section 5.2.3 describes candidate rules for adding and dropping layers. A performance study of the resulting policy is presented in Section 5.2.4.

5.2.1 Efficient Work-ahead

Note that in periodic broadcast systems, each segment transmission is at a fixed rate, and is received by many clients. It is not feasible to temporarily increase the transmission rate so that some particular client can achieve work-ahead. Thus, the only efficient technique for performing work-ahead is to start listening to segment transmissions earlier than what would be dictated by the protocol.

Accomplishing work-ahead in this manner is complicated by the cyclic transmission of segments. As an illustration, consider a client that has partially received a segment, and then stops listening to the channel broadcasting this segment owing to a drop in the available bandwidth. If the client later resumes reception on the channel in time to receive data equivalent in amount to the data it is missing, it will be able to receive the remainder of the segment only if its reception period aligns with the broadcast of the missing portion.

A remedy to the above problem is to apply erasure codes to each segment so that a channel transmits a very long sequence of encoded packets instead of transmitting the unencoded packets in a cyclic fashion. With erasure codes, essentially, all packets are equivalent, and a segment can be reconstructed from any subset of packets equal (or possibly slightly longer) in total size to the size of the segment. The RPB protocols developed in Chapter 3 used erasure codes to enable efficient packet loss recovery. The techniques developed here can be applied with these protocols, as well as with other protocols such as the Optimized HPB protocols in which each segment is received in its entirety before its playback commences, and thus can be erasure-coded[3].

5.2.2 Policy for Work-ahead Allocation

The work-ahead policy determines how the extra bandwidth available (in addition to that needed for the current layer subscription) is allocated among the layers to provide maximal protection against short-term bandwidth drops. Two main considerations can be identified. First, as described in [104], work-ahead on lower layers is "safer" since it reduces the chances of work-ahead data being wasted in the event of a layer drop. To understand why it is "safer" to work-ahead on lower layers, note that in the event of a layer drop, any work-ahead on that layer is lost. Second,

[3]Protocols that require clients to listen on all channels, such as Harmonic Broadcasting and its variants, may not be able to perform work-ahead without using additional server channels to read-ahead the required data.

spreading work-ahead among many layers allows a greater short-term reception rate reduction. This is because, regardless of the amount of work-ahead available on a layer, the bandwidth consumption of that layer can only be reduced to zero.

Another important consideration is the granularity at which work-ahead should be performed. Note that transmission of erasure-coded segments implies that an entire segment must be received in order to begin segment playback. Therefore, the client does not start reading segment $i + 1$ in layer j unless it is reading or has read all eligible earlier segments. This constraint avoids making frequent changes to multicast group memberships.

In view of the above considerations, and following considerable experimentation, the following rules are proposed:

- All subscribed layers *except* the topmost of the subscribed layers are eligible for work-ahead. The topmost layer is eligible for work-ahead when the available bandwidth cannot be fully used otherwise.

- Among the layers eligible for work-ahead, excess bandwidth is allocated to early reception of segments in an order corresponding to their reception deadlines; in case of a tie, preference is given to the segments from the lowest layer.

5.2.3 Policy for Adding/Dropping Layers

The decision to add a layer may depend on currently available bandwidth, currently achieved work-ahead, the bandwidth requirements of the currently subscribed layers and the new layer, and an estimate of the bandwidth available in the future. Obtaining a reasonable estimate of the future available bandwidth may, however, be quite difficult. Thus, in previous work [104] and in this work, the decision to add a new layer is taken when the instantaneous bandwidth exceeds the bandwidth requirement of the subscribed layers in addition to the new layer, provided some work-ahead condition is satisfied.

The aim of the work-ahead condition is to ensure that all existing layers plus the new layer can be maintained in the event of a short-term bandwidth drop immediately following a layer addition. Note that if the average available bandwidth over some fairly long window of time W exceeds the bandwidth required to sustain the currently subscribed layers and the new layer, and the client has been accumulating work-ahead at the maximum rate, then enough work-ahead must already be available to sustain a single layer drop for the same-sized time window. Combining this work-ahead condition with the instantaneous bandwidth requirement, the following rule for adding a layer can be obtained:

- Add a layer if both the available average bandwidth over the last W seconds and the instantaneous available bandwidth are greater than $(1 + \epsilon)$ times the sum of the bandwidth requirements of the currently subscribed layers and the candidate new layer, where ϵ is a small constant used for removing hysteresis (e.g., $\epsilon = 0.05$).

Determining the bandwidth required for the current subscription level needs some special consideration. Note that unlike unicast streaming, the bandwidth requirement in a periodic broadcast system does not equal the layer bit rate as the clients are required to listen on multiple server channels at an aggregate rate greater than the layer bit rate. In the context of an Optimized HPB system delivering media layer l, the bandwidth $b_{j,l}$ required by a client that starts receiving a segment j broadcast of that layer (i.e., segment j is the first segment from layer l that can potentially be played back at the client) can be easily computed. Note that $b_{j,l}$ depends on the reception deadlines for segments $k \geq j$.

With Optimized HPB systems, a client can choose a higher start-up delay that enables it to subscribe to more layers, using a lower total reception rate on each layer. This lower reception rate is possible because of the previously accumulated work-ahead. Let b_l^d denote the required client bandwidth for layer l when the layer is joined at the first segment, for a start-up delay d, as determined for a particular Optimized HPB protocol setting (i.e., as described in Section 5.1.2).

The quantities b_l^d and $b_{j,l}$ are used to determine the bandwidth requirement of a layer, when determining whether or not to add a new layer, as follows:

- During the client start-up delay, the bandwidth requirement of a layer is taken as b_l^d. This is the minimum bandwidth required to start playback on l layers after start-up delay d.

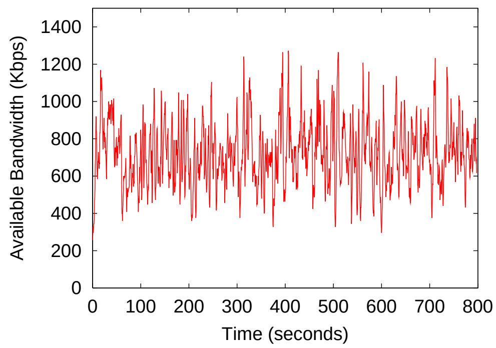
- Following the start-up delay, for each layer that was subscribed during start-up, the bandwidth requirement is taken as $\min\left(b_l^d, b_{j,l}\right)$. This constraint considers the fact that with Optimized HPB $b_{j,l}$ starts to decrease well before the end of the media file. For each other layer, the bandwidth requirement is taken as $b_{j,l}$.

The policy for dropping layers is very simple. A client drops the highest currently subscribed layer if and only if there is insufficient available bandwidth to receive all the segments that the periodic broadcast protocol requires at that point in time.

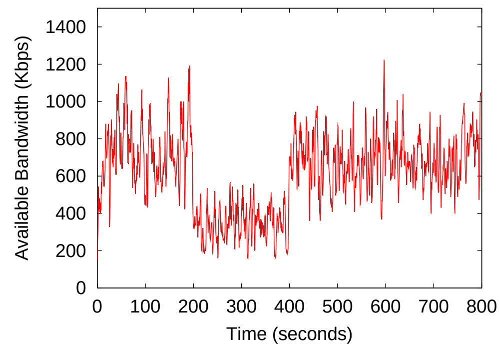
5.2.4 Performance Evaluation

Evaluation of the proposed dynamic quality adaptation mechanisms requires, as input, a trace of time-varying available bandwidth. Note that the rate control protocols proposed in the literature and in the previous chapter could be used to track available bandwidth, but this chapter focuses on the quality adaptation component exclusively. An integrated study of rate and quality adaptation is a fruitful avenue for future work.

The proposed dynamic quality adaptation mechanism is evaluated using a variety of traces of available bandwidth as determined using TFRC [45], a recently proposed *unicast* TCP-friendly congestion control scheme for streaming media applications, as obtained using the ns-2 simulator. Representative results for two such traces are shown here. In the first trace, illustrated in Figure 5.6(a), there are no long-term fluctuations in the available bandwidth. This trace reflects the bandwidth of a TFRC flow when it is competing with 8 TCP/Sack flows and 7 other TFRC flows



(a) No Long-Term Bandwidth Fluctuations



(b) Long-Term Bandwidth Fluctuation

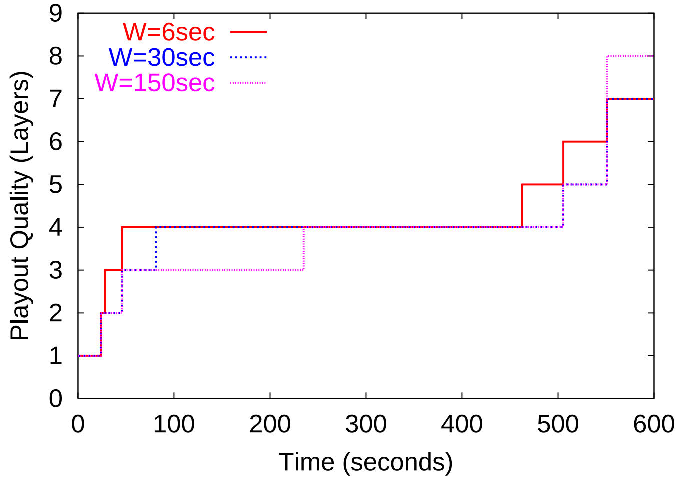
Figure 5.6: Sample TFRC Traces

on a 16 Mbps bottleneck link. To eliminate possible phase effects, 4 TCP/Sack flows are used in the reverse direction. The round-trip propagation delay between each source/destination pair is 40 milliseconds. The set-up assumes RED [48] queue management with buffer capacity of 160 packets.[4] The second trace, illustrated in Figure 5.6(b), is obtained using the same simulation set-up, except that a 4Mbps CBR flow of duration 200 seconds is introduced at time 200. This trace is used to illustrate the responsiveness of the proposed policies to longer-term changes in available bandwidth.

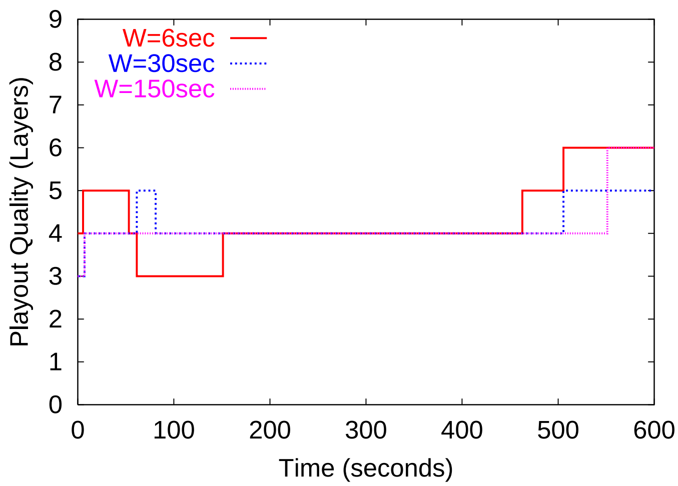
The sample evaluations of the quality adaptation mechanisms using the above traces assume that: 1) the media file being streamed is of duration 10 minutes; 2) the media file consists of 10 layers, each with bit rate 128Kbps; and 3) each layer uses Optimized HPB with $r = 0.25$ and $K = 40$. The parameter ϵ in the quality adaptation policy is fixed at 0.05 (i.e., 5%).

Figure 5.7 shows how short-term variation in available bandwidth is effectively absorbed by the adaptation policy for different delay/quality tradeoffs. Time is measured in this figure and Figure 5.9 from the beginning of playout, not the beginning of reception which is chosen as the beginning of the respective trace in Figure 5.6.

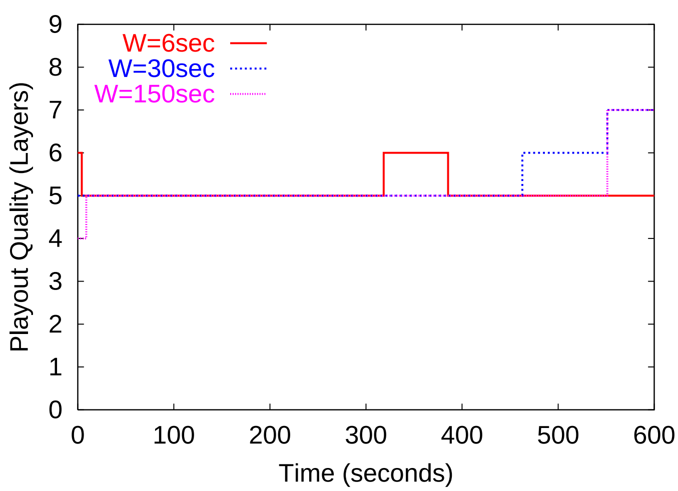
[4]RED maintains a weighted average of the queue length, and uses this measure to probabilistically drop packets even before the queue is full. The drop probability increases from 0 to a maximum drop probability as the average queue length increases from *thresh* to *max_thresh*. All packets are of size 1 KB. Following guidelines in previous work, *thresh* and *max_thresh* are set to 32 and 128 packets, respectively. The simulations use the default ns-2 values for all other RED parameters.



(a) Start-up Delay = 3 seconds



(b) Start-up Delay = 20 seconds



(c) Start-up Delay = 120 seconds

Figure 5.7: Playout Quality with Quality Adaptation Mechanisms
(Figure 5.6(a) available bandwidth trace: no long-term bandwidth fluctuation)

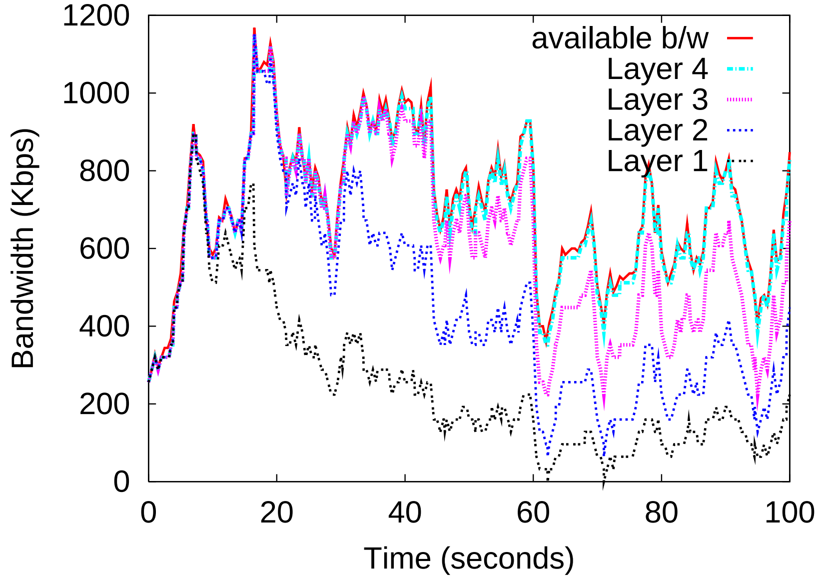
With the trace in Figure 5.6(a), the client has, on average, an achievable data rate of about 700Kbps. That is, the average client data rate (in units of media playback rate) b is 5.5, and therefore, from the discussion in Section 5.1.2, it is expected that start-up delays of 3 seconds, 20 seconds, and 120 seconds, will map to media qualities of 2 layers, 4 layers, and 5 layers, respectively. The simulation results in Figure 5.7 show that the playout media quality is as expected or higher for almost the entire playback duration. Playback quality increases towards the end of the media file as the client bandwidth requirement with Optimized HPB protocols decreases for later portions of the media file (see the discussion of the $b_{j,l}$ parameter in Section 5.2.3). The increase in subscription for the playback of the end portion of the media file is also enabled by the work-ahead that has been accumulated by this point on the lower layers. The results also illustrate the impact of the averaging window W. Generally, long windows result in more work-ahead, and therefore permits higher quality in the later portion of the playback, at the possible cost of lower quality in the initial portion.

Figure 5.8 illustrates how bandwidth fluctuations are absorbed by work-ahead. The curve for a layer indicates the bandwidth allocated to that layer and all lower layers at the indicated point in time. Note that the total bandwidth consumption closely tracks the available bandwidth curve.

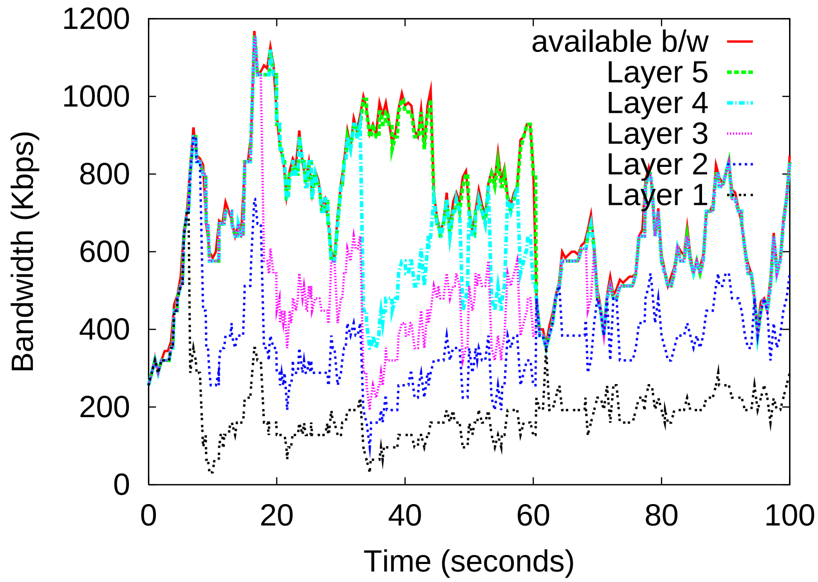
Figure 5.9 shows the results for the trace of Figure 7(b). Although some fluctuation in playback quality is unavoidable in this case, these fluctuations are relatively modest, compared to the fluctuations in available bandwidth.

5.3 Bandwidth Skimming for Heterogeneous Clients

This section describes how bandwidth skimming can be extended to support heterogeneous clients, and then briefly discusses work-ahead and quality adaptation. Unlike work-ahead for periodic broadcast discussed in the previous section, there are server bandwidth costs for implementing work-ahead mechanisms.

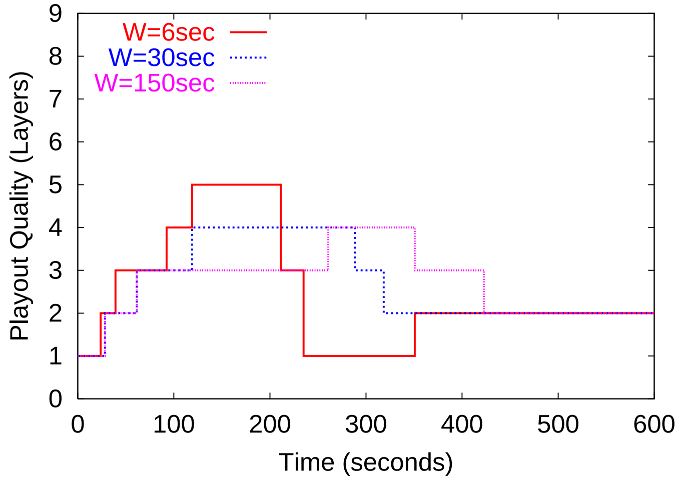


(a) Start-up Delay = 3 seconds

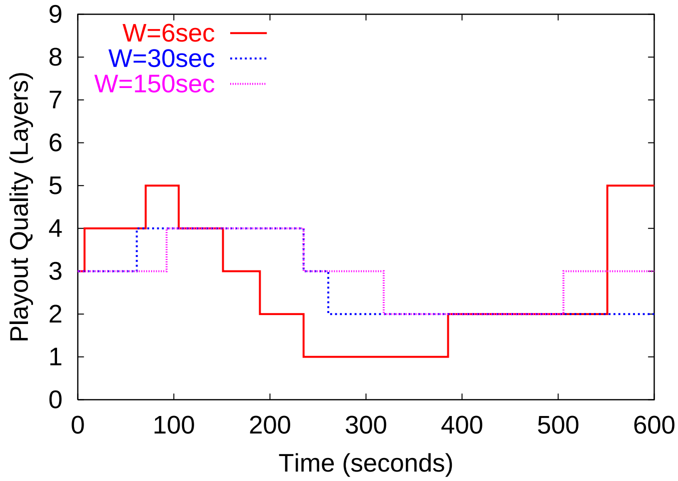


(b) Start-up Delay = 120 seconds

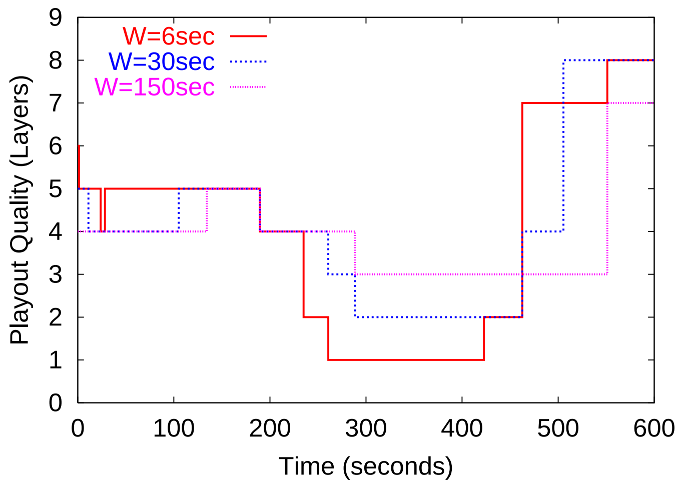
Figure 5.8: Cumulative Bandwidth Allocation with Quality Adaptation Mechanisms (Initial portion of Figure 5.6(a) available bandwidth trace)



(a) Start-up Delay = 3 seconds



(b) Start-up Delay = 20 seconds



(c) Start-up Delay = 120 seconds

Figure 5.9: Playout Quality with Quality Adaptation Mechanisms
(Figure 5.6(b) available bandwidth trace: single long-term bandwidth fluctuation)

5.3.1 Supporting Heterogeneity

This section extends the bandwidth skimming protocols to support clients with differing achievable client reception rates, for a single monolithic media object or a single layer of a media file. The Latest Patch and Partition stream merging policies are considered here. Tradeoffs between start-up delay and quality are possible in a hybrid batching/bandwidth skimming [40] system; however, this study focuses on pure bandwidth skimming systems.

In the case of Partition, clients merge with earlier groups by listening to successively larger numbers of substreams of the group being merged with, while listening to fewer of their own group's substreams. An implementation may have substreams being transmitted on different multicast addresses at fixed rates, with clients joining/leaving these multicast groups depending on the available bandwidth, possibly determined using a TCP friendly multirate congestion control protocol, such as the VMRC protocol proposed in the previous chapter.

The Latest Patch policy requires clients to receive a single media stream at a higher rate than the media playback rate. Specifically, clients advance at the rate of the slowest client in their group, attempting to merge with an earlier group. The rate of the group can be chosen as the minimum achievable data rate among the clients in the group, as might be available on using a single rate multicast rate control policy such as PGMCC [109] or TFMCC [132].

In this subsection, only the case of clients with static (possibly heterogeneous) available bandwidth is considered. Figure 5.10 shows the required server bandwidth for the Latest Patch protocol and homogeneous system in which clients have available bandwidth 1.25 or 2 times the media playback bit rate, and for a heterogeneous system with a mix of these two types of clients. These results were obtained using simulation assuming Poisson request arrivals. As illustrated in the figure, at relatively low client request rates the required server bandwidth for delivery of a file is influenced most by the most common client bandwidth, while at high client request rates, the clients with the lowest available bandwidth have a relatively larger impact.

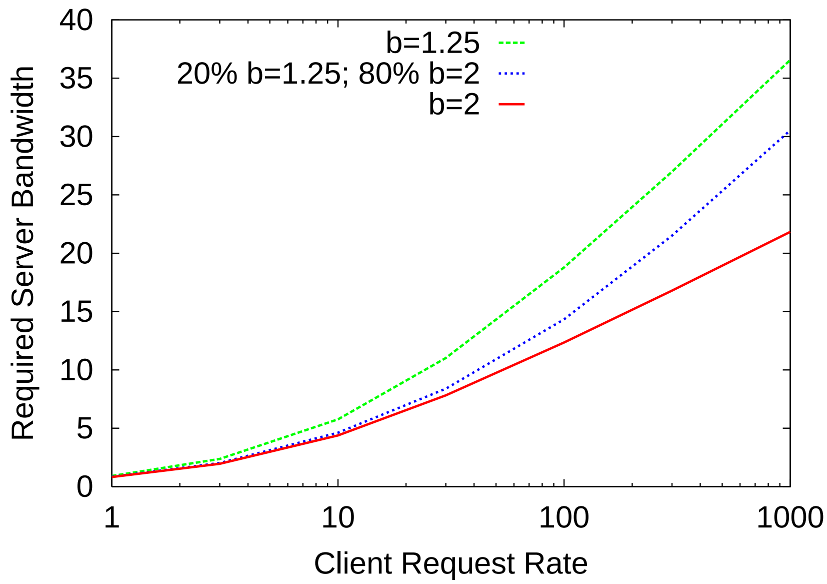


Figure 5.10: Performance of Bandwidth Skimming with Heterogeneous Clients

5.3.2 Dynamic Quality Adaptation

Consider the delivery of a layered media file using bandwidth skimming, with each layer of the media file being delivered by a separate instance of the protocol. Each client can determine the number of layers it can receive based on its available bandwidth. Dynamic quality adaptation would require mechanisms that define how work-ahead should be allocated among the layers, and thus the rate of the corresponding streams, given the current available bandwidth. Note that the Latest Patch, by design, is required to perform work-ahead by receiving data at a higher rate than the layer bit rate to enable stream merging, and may be more suitable than Partition which allows bandwidth changes only at the granularity of substream transmission rates.

Dynamic quality adaptation is complicated by the fact that all clients in a group share the same multicast stream, and therefore, must progress at the same rate. Layer subscription and work-ahead allocation policies must base their decision on the needs of the individual clients, and at the same time attempt to keep clients of a group aggregated together. In addition, quality adaptation policies must consider how the mechanisms of the rate control policy influence quality adaptation decisions. In Chapter 6, possible dynamic quality adaptation approaches for bandwidth skimming, that are currently being investigated, are outlined.

5.4 Summary

This chapter developed static and dynamic quality adaptation mechanisms for scalable on-demand streaming to heterogeneous clients. In static quality adaptation, enough work-ahead is accumulated before playback to support streaming higher quality media than would be otherwise possible. Previously proposed periodic broadcasts are designed for a single homogeneous client data rate. The new Optimized HBP protocols developed here efficiently support clients within a specified range of achievable data rates, and also allow better tradeoffs between start-up delay and media quality. A complementary dynamic quality adaptation mechanism is developed and evaluated that performs work-ahead during playback to provide more uniform playback quality, given time varying available bandwidth.

Chapter 6

Conclusions

To conclude this work, this chapter summarizes the thesis, enumerates the main contributions, and briefly outlines future work directions.

6.1 Thesis Summary

Chapters 1 and 2 described the goals of the thesis research and relevant prior work.

Chapter 3 developed and evaluated the new periodic broadcast and bandwidth skimming protocols, namely Reliable Periodic Broadcast (RPB) and Reliable Bandwidth Skimming (RBS), that allow clients to recover from packet loss. New scalability bounds for both immediate service protocols and bounded delay protocols, assuming alternative packet loss recovery techniques, were presented. The performance of these protocols was compared with that of the bounds, and the results suggest that the new protocols can achieve close to the best possible server bandwidth scalability for a given set of client characteristics. An implementation of the RBS protocol is presented as "proof of concept" of the proposed protocols.

Chapter 4 developed a new equation-based multirate congestion control protocol called Vegas Multicast Rate Control (VMRC) that can be used in conjunction with the scalable streaming protocols developed in this research. The VMRC protocol uses a recently proposed throughput model for TCP Vegas. This protocol has the key advantage of operating without inducing packet loss when the bottleneck link is lightly loaded. The VMRC protocol incorporates a new technique for dynamically adjusting the TCP Vegas threshold parameters based on measured characteristics of the network. This technique implements fair sharing of network resources with other types of competing flows, including widely deployed versions of TCP such as TCP Reno, which is not possible with the previously defined static Vegas threshold parameters. This technique might be fruitfully incorporated in TCP Vegas itself to aid in its incremental deployment.

Chapter 4 also presented a detailed performance evaluation study of VMRC. The performance of VMRC is compared with that of an analogous protocol that is based on a TCP Reno throughput model to highlight the benefits of Vegas-like rate control. Furthermore, the design of VMRC is evaluated along the key dimensions of synchronization policy, round-trip time and propagation delay measurement techniques, data transmission policy, and protocol reactivity.

Chapter 5 discussed quality adaptation techniques for maximizing playback quality given a certain available client reception bandwidth. Note that the RPB protocol was designed assuming a single homogeneous client reception rate. The Optimized Heterogeneous Periodic Broadcast (HPB) protocols developed in this chapter efficiently support clients within a specified range of data rates. Using this protocol, static quality adaptation in the form of tradeoffs between start-up delay and media quality are possible. Chapter 5 also developed and evaluated a mechanism for dynamic quality adaptation using work-ahead during playback that can provide more uniform playback quality given time-varying available bandwidth due to rate adaptation (using a protocol like VMRC).

6.2 Contributions

The following are the main contributions of this work:

- Development of the Reliable Periodic Broadcast (RPB) protocol family that provides scalable on-demand streaming with packet loss recovery. Recall that previous periodic broadcast protocols do not provide scalable loss recovery.

- The RPB protocols also improve upon the previous periodic broadcast protocols by allowing the aggregate reception rate of each client to be only a small fraction greater than the media playback rate. Previous periodic broadcast protocols required transmission rate to the clients to be at least twice the media playback rate. Therefore, the new protocols allow delivery of higher quality media than would be possible with these prior protocols.

- New Reliable Bandwidth Skimming (RBS) protocols are developed that extend previous bandwidth skimming protocols to efficiently provide packet loss recovery.

- Scalability bounds are developed, assuming alternative packet loss recovery techniques, for both bandwidth skimming and periodic broadcast protocols. Results from these bounds suggest that the RPB and RBS protocols achieve nearly the best possible server bandwidth scalability for an assumed set of client characteristics.

- The RBS protocol has been implemented in the SWORD prototype. This implementation is a "proof of concept" of the protocols presented in this work. Preliminary experiments with the prototype have demonstrated the correctness of the implementation.

- A new equation-based multirate congestion control protocol called Vegas Multicast Rate Control (VMRC) has been developed and extensively evaluated. The VMRC protocol utilizes a recently proposed TCP Vegas throughput model to determine the appropriate reception rate at each client. The VMRC protocol has the advantage of providing a throughput share that is less oscillatory than that possible with protocols that are based on TCP Reno. Furthermore, the protocol operates without inducing packet losses when the bottleneck link is lightly loaded.

- The VMRC protocol incorporates a new technique for dynamically adjusting the TCP Vegas threshold parameters based on measured characteristics of the network. This technique implements fair sharing of network resources with other types of competing flows, including widely deployed versions of TCP such as TCP Reno, which is not possible with the previously defined static Vegas threshold parameters.

- New Heterogeneous Periodic Broadcast (HPB) protocols are developed that extend the RPB protocols to efficiently support heterogeneous clients. Efficient static quality adaptation mechanisms are proposed, in the context of the HPB protocols, that allow tradeoffs between start-up delay and media quality.

- Efficient dynamic quality adaptation techniques are also developed for HPB and RPB systems that effectively provide more uniform playback quality given time varying available bandwidth to clients (as would be obtained using a rate control policy).

6.3 Future Work

This thesis has addressed several important issues pertaining to current and future video-on-demand systems. In this section, related open areas of research are discussed that present fruitful avenues for future work:

- Supporting interactive client requests such as fast forward, rewind, and pause appear to pose challenges in the context of RPB systems. If the client does not discard segments it has already played back, the rewind operation is easy to implement. The pause operation may be implemented by continuing to receive segments and buffering this data while waiting for the client to resume playback. However, there might be client side storage limitations, in which case special techniques are required to implement these two interactive functions. Supporting fast forward operation presents further challenges. Note that a limited form of fast forward may be possible owing to the "work-ahead" nature of the protocol. That is, if some segments are already available in the client's buffer, fast forward operation requires playing selective frames from the available segments. When segments are not available in the client's buffer, fast forward may entail the server sending an unicast stream with the required data.

- Considerable insight can be gained by virtue of prototype implementation and experimentation. Ongoing efforts include conducting local area and wide area experiments with the prototype. These experiments are expected to provide a better understanding of the impact of multicast join/leave latencies, quantify the overhead of merging algorithms, identify characteristics of packet loss, and evaluate packet loss recovery performance.

- An ongoing problem has been the lack of a dependable multicast service. This has motivated development of application-level multicast support within SWORD. An interesting future work direction will be to evaluate performance of the on-demand streaming protocol in an application-level multicast setting.

- The VMRC performance study assumed a layered media encoding. The VMRC protocol should also be applicable in the context of non-layered media encodings, where different clients dynamically select among different monolithically encoded object versions.

- The performance study of VMRC suggests that dynamically setting the TCP Vegas threshold can substantially improve fairness of a TCP Vegas flow when it competes with more aggressive flows such as TCP Reno. As part of future work, this dynamic threshold estimation technique will be implemented in the TCP Vegas module of the *ns-2* simulator, and a detailed performance study will be conducted to determine the benefits of this approach.

- The dynamic quality adaptation mechanism developed in this work considered the rate adaptation module to be independent from the quality adaptation module. However, the rate and quality adaptation components could cooperate to allow for more efficient use of available bandwidth. An integrated study of rate and quality adaptation using RPB/HPB will be undertaken as future work.

- A layered media file may be delivered using the bandwidth skimming protocol, with each layer of the media file being delivered using a separate instance of the protocol. Each client can determine the number of layers it can receive based on its available bandwidth. Dynamic quality adaptation would require mechanisms that define how work-ahead should be allocated given the current available bandwidth. Dynamic quality adaptation is complicated by the fact that all clients in a group share the same multicast stream, and therefore, must progress at the same rate. Hence, work-ahead policies must consider both the needs of the individual clients as well as the requirements of the group.

- The evaluation of the dynamic quality adaptation policy relied on a single metric, namely the subscription level of the client. However, this metric may not adequately reflect the *perceived* quality of playback at the client. Ascertaining the perceived quality will involve human subjects, thus indicating that collaboration with researchers in the area of human computer interaction may be necessary.

References

[1] C. Aggarwal, J. Wolf, and P. Yu. Design and Analysis of Permutation-Based Pyramid Broadcasting. *Multimedia Systems*, 7(6):439–448, 1999.

[2] C. Aggrawal, J. Wolf, and P. Yu. A Permutation-Based Pyramid Broadcasting Scheme for Video-on-Demand Systems. In *Proc. IEEE ICMCS '96*, Hiroshima, Japan, June 1996.

[3] C. Aggrawal, J. Wolf, and P. Yu. On Optimal Batching Policies for Video-on-Demand Storage Servers. In *Proc. IEEE ICMCS '96*, Hiroshima, Japan, June 1996.

[4] C. Aggrawal, J. Wolf, and P. Yu. On Optimal Piggyback Merging Policies for Video-on-Demand Systems. In *Proc. ACM SIGMETRICS '96*, Philadelphia, PA, May 1996.

[5] J. Ahn, P. Danzig, Z. Liu, and L. Yan. Evaluation of TCP Vegas: Emulation and Experiment. *IEEE Trans. on Communications*, 25(4):185–195, October 1995.

[6] M. Allman. A Web Server's View of the Transport Layer. *ACM Computer Communication Review*, 30(5):10–20, October 2000.

[7] J. Almeida, D. Eager, M. Ferris, and M. Vernon. Provisioning Content Distribution Networks for Streaming Media. In *Proc. IEEE INFOCOM '02*, New York, NY, June 2002.

[8] J. Almeida, J. Krueger, D. Eager, and M. Vernon. Analysis of Educational Media Server Workloads. In *Proc. NOSSDAV '01*, Port Jefferson, NY, June 2001.

[9] K. Almeroth. The Evolution of Multicast: From the MBone to Interdomain Multicast to Internet2 Deployment. *IEEE Network*, 14(1):10–20, January/February 2000.

[10] M. Ammar and J. Wong. On the Optimality of Cyclic Transmission in Teletext Systems. *IEEE Trans. on Communications*, 7(6):68–73, January 1987.

[11] M. Arlitt and C. Williamson. Internet Web Servers: Workload Characterization and Performance Implications. *IEEE/ACM Trans. on Networking*, 5(5):631–645, October 1997.

[12] T. Ballardie, P. Francis, and J. Crowcroft. Core Based Trees (CBT): An Architecture for Scalable Multicast Routing. In *Proc. ACM SIGCOMM '95*, San Francisco, CA, September 1995.

[13] D. Bansal and H. Balakrishnan. Binomial Congestion Control Algorithms. In *Proc. IEEE INFOCOM '01*, Anchorage, AK, April 2001.

[14] C. Barakat. TCP/IP Modeling and Validation. *IEEE Network*, 15(3):38–47, May/June 2001.

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

[16] Y. Birk and R. Mondri. Tailored Transmissions for Efficient Near-Video-On-Demand Service. In *Proc. IEEE ICMCS '99*, Florence, Italy, June 1999.

[17] J. Bolot, S. Parisis, and D. Towsley. Adaptive FEC-Based Error Control for Internet Telephony. In *Proc. IEEE INFOCOM '99*, New York, NY, March 1999.

[18] J. Boyce and R. Gaglianello. Packet Loss Effects on MPEG Video Sent Over the Public Internet. In *Proc. ACM Multimedia '00*, Bristol, UK, September 2000.

[19] B. Braden, D. Clark, J. Crowcroft, B. Davis, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang. Recommmendations on Queue Management and Congestion Avoidance in the Internet. RFC 2309, April 1998.

[20] L. Brakmo, S. O'Malley, and L. Peterson. TCP Vegas: New Techniques for Congestion Detection and Avoidance. In *Proc. ACM SIGCOMM '94*, London, UK, September 1994.

[21] L. Bramko and L. Peterson. TCP Vegas: End-to-End Congestion Avoidance on a Global Internet. *IEEE Journal on Selected Areas in Communications*, 13(8):1465–1480, October 1995.

[22] J. Byers, M. Frumin, G. Horn, M. Luby, M. Mitzenmacher, A. Roetter, and W. Shaver. FLID-DL: Congestion Control for Layered Multicast. In *Proc. NGC '00*, Stanford, CA, November 2000.

[23] J. Byers and G. Kwon. STAIR: A Practical AMID Multirate Multicast Congestion Control. In *Proc. NGC '01*, London, UK, November 2001.

[24] J. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A Digital Fountain Approach to Reliable Distribution of Bulk Data. In *Proc. ACM SIGCOMM '98*, Vancouver, Canada, September 1998.

[25] Y. Cai, K. Hua, and K. Vu. Optimized Patching Performance. In *Proc. MMCN '99*, San Jose, CA, January 1999.

[26] G. Carle and E. W. Biersack. Survey of Error Recovery Techniques for IP-based Audio-Visual Multicast Applications. *IEEE Network*, 11(6):24–36, November/December 1997.

[27] S. Carter and D. Long. Improving Video-on-Demand Server Efficiency Through Stream Tapping. In *Proc. ICCCN '97*, Las Vegas, NV, September 1997.

[28] M. Chesire, A. Wolman, G. Voelker, and H. Levy. Measurement and Analysis of a Streaming-Media Workload. In *Proc. USITS '01*, San Francisco, CA, March 2001.

[29] S. Cheung, M. Ammar, and X. Li. On the use of Destination Set Grouping to Improve Fairness in Multicast Video Distribution. In *Proc. IEEE INFOCOM '96*, San Francisco, CA, March 1996.

[30] P. Chou, A. Mohr, A. Wang, and S. Mehrotra. FEC and Pseudo-ARQ for Receiver-driven Layered Multicast of Audio and Video. In *Proc. IEEE Data Compression Conf.*, Snowbird, UT, March 2000.

[31] Y. Chu, S. Rao, and H. Zhang. A Case for End System Multicast. In *Proc. ACM SIGMETRICS '00*, Santa Clara, CA, June 2000.

[32] A. Dan, D. Sitaram, and P. Shahabuddin. Scheduling Policies for an On-Demand Video Server. In *Proc. ACM MULTIMEDIA '94*, San Francisco, CA, October 1994.

[33] S. Deering. *Multicast Routing in a Datagram Internetwork*. PhD thesis, Dept. of Computer Science, Stanford University, 1991.

[34] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, A. Helny, D. Meyers, and L. Wei. Protocol Independent Multicast Version 2 Specification. Internet Draft, August 1998.

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

[36] C. Diot, B. Levine, B. Lyles, H. Kassan, and D. Balsiefien. Deployment Issues for the IP Multicast Service and Architecture. *IEEE Network*, 14(1):10–20, January/February 2000.

[37] H. Dykeman, M. Ammar, and J. Wong. Scheduling Algorithms for Videotex Systems under Broadcast Delivery. In *Proc. ICC '86*, Toronto, Canada, June 1986.

[38] D. Eager and M. Vernon. Dynamic Skyscraper Broadcasts for Video-on-Demand. In *Proc. MIS '98*, Istanbul, Turkey, September 1998.

[39] D. Eager, M. Vernon, and J. Zahorjan. Optimal and Efficient Merging Schedules for Video-on-Demand Servers. In *Proc. ACM MULTIMEDIA '99*, Orlando, FL, November 1999.

[40] D. Eager, M. Vernon, and J. Zahorjan. Bandwidth Skimming: A Technique for Cost-Effective Video-on-Demand. In *Proc. MMCN '00*, San Jose, CA, January 2000.

[41] D. Eager, M. Vernon, and J. Zahorjan. Minimizing Bandwidth Requirements for On-Demand Data Delivery. *IEEE Trans. on Knowledge and Data Engineering*, 13(5):742–757, September/October 2001.

[42] K. Fall and S. Floyd. Congestion Avoidance and Control. *Computer Communication Review*, 26(3):5–21, July 1996.

[43] W. Fenner. Internet Group Management Protocol, Version 2. RFC 2236, November 1997.

[44] S. Floyd and K. Fall. Promoting the Use of End-to-End Congestion Control in the Internet. *IEEE/ACM Trans. on Networking*, 7(4):458–472, 1999.

[45] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based Congestion Control for Unicast Applications. In *Proc. ACM SIGCOMM '00*, Stockholm, Sweden, August 2000.

[46] S. Floyd and T. Henderson. The Newreno Modification to TCP's Fast Recovery Algorithm. RFC 2582, April 1999.

[47] S. Floyd and V. Jacobson. On Traffic Phase Effects in Packet-Switched Gateways. *Internetworking: Research and Experience*, 3(3):115–156, September 1992.

[48] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Trans. Networking*, 1(4):397–413, August 1993.

[49] S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang. A Reliable Multicast Framework for Light-Weight Sessions and Application Level Framing. *IEEE/ACM Trans. on Networking*, 5(6):784–803, December 1997.

[50] S. Floyd, V. Jacobson, S. McCanne, C. Liu, and L. Zhang. A Reliable Multicast Framework for Light-Weight Sessions and Application Level Framing. In *Proc. ACM SIGCOMM '95*, Boston, MA, August 1995.

[51] S. Floyd and E. Kohler. Internet Research Needs Better Models. In *Proc. of First Workshop on Hot Topics in Networking*, Princeton, NJ, October 2002.

[52] S. Fredj, T. Bonald, A. Proutiere, G. Regnie, and J. Roberts. Statistical Bandwidth Sharing: A Study of Congestion at Flow Level. In *Proc. ACM SIGCOMM '01*, San Diego, CA, August 2001.

[53] D. L. Gall. MPEG: A Video Compression Standard for Multimedia Applications. *Communications of ACM*, 34(4):46–58, April 1991.

[54] L. Gao, J. Kurose, and D. Towsley. Efficient Schemes for Broadcasting Popular Videos. In *Proc. NOSSDAV '98*, Cambridge, UK, July 1998.

[55] L. Gao and D. Towsley. Supplying Instantaneous Video-on-Demand Services Using Controlled Multicast. In *Proc. ICMCS '99*, Florence, Italy, June 1999.

[56] L. Golubchik, J. Lui, and R. Muntz. Reducing I/O Demand in Video-on-Demand Storage Servers. In *Proc. ACM SIGMETRICS '95*, Ottawa, Canada, May 1995.

[57] M. Goyal, R. Guerin, and R. Rajan. Predicting TCP Throughput from Non-Invasive Network Sampling. In *Proc. IEEE INFOCOM '02*, Hiroshima, Japan, March 2002.

[58] H. Holbrook and D. Cheriton. IP Multicast Channels: EXPRESS Support for Large-scale Single-source Applications. In *Proc. ACM SIGCOMM '99*, Cambridge, MA, August 1999.

[59] A. Hu. Video-on-Demand Broadcasting Protocols: A Comprehensive Study. In *Proc. IEEE INFOCOM '01*, Anchorage, AK, April 2001.

[60] A. Hu, I. Nikolaidis, and P. Beek. On the Design of Efficient Video-on-Demand Broadcast Schemes. In *Proc. MASCOTS '99*, Marchland, MD, October 1999.

[61] K. Hua, Y. Cai, and S. Sheu. Patching: A Multicast Technique for True Video-on-Demand. In *Proc. ACM MULTIMEDIA '98*, Bristol, UK, September 1998.

[62] K. Hua and S. Sheu. Skyscraper Broadcasting: A New Broadcasting Scheme for Metropolitan Video-on-Demand Systems. In *Proc. ACM SIGCOMM '97*, Cannes, France, September 1997.

[63] V. Jacobson. Congestion Avoidance and Control. *Computer Communication Review*, 18(4):314–329, August 1988.

[64] V. Jacobson. Modified TCP Congestion Avoidance Algorithm. Email to end-to-end interest mailing list, April 1990.

[65] J. Jannotti, D. Gifford, K. Johnson, M. Kaashoek, and J. O'Toole. Overcast: Reliable multicasting with an overlay network. In *Proc. USENIX OSDI '00*, San Diego, CA, October 2000.

[66] H. Jiang and C. Dovorolis. Passive Estimation of TCP Round-trip Times. *ACM Computer Communication Review*, 32(3):75–88, July 2002.

[67] W. Jiang and H. Schulzrinne. Modeling of Packet Loss and Delay and their Effect on Real-Time Multimedia Service Quality. In *Proc. NOSSDAV '00*, Chapel Hill, NC, June 2000.

[68] S. Jin and A. Bestavros. Scalability of Multicast Delivery for Non-sequential Streaming Access. In *Proc. ACM SIGMETRICS '02*, Marina Del Rey, CA, June 2002.

[69] L. Juhn and L. Tseng. Fast Data Broadcasting and Receiving Scheme for Popular Video Service. *IEEE Trans. on Broadcasting*, 44(1):100–105, March 1998.

[70] T. Kuang and C. Williamson. A Measurement Study of Real Media Audio/Video Streaming Traffic. In *Proc. SPIE ITCOM '02*, Boston, MA, July 2002.

[71] J. Kurose and K. Ross. *Computer Networking: A Top-Down Approach Featuring the Internet*. Addison-Wesley Longman, Inc., 2003.

[72] T. Laksham and U. Madhow. The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss. *IEEE/ACM Trans. on Networking*, 5(3):336–360, July 1997.

[73] A. Legout and E. Biersack. Pathological Behaviors for RLM and RLC. In *Proc. NOSSDAV'00*, Chapel Hill, NC, June 2000.

[74] A. Legout and E. Biersack. PLM: Fast Convergence for Cumulative Layered Multicast Transmission Schemes. In *Proc. ACM SIGMETRICS '00*, Santa Clara, CA, June 2000.

[75] X. Li, M. H. Ammar, and S. Paul. Video Multicast over the Internet. *IEEE Network*, 13(2):46–60, April 1999.

[76] X. Li, S. Paul, and M. Ammar. Multi-Session Rate Control for Layered Video Multicast. In *Proc. MMCN'99*, San Jose, CA, January 1999.

[77] X. Li, S. Paul, P. Pancha, and M. Ammar. Layered Video Multicast with Retransmission (LVMR): Evaluation of Error Recovery Schemes. In *Proc. NOSSDAV '97*, St. Louis, MO, May 1997.

[78] Y. Liang, E. Steinbach, and B. Girod. Real-time Voice Communication over the Internet Using Packet Path Diversity. In *Proc. ACM MULTIMEDIA '01*, Ottawa, Canada, December 2001.

[79] J. Lin and S. Paul. RMTP:A Reliable Multicast Transport Protocol. In *Proc. IEEE INFOCOM '96*, San Francisco, CA, March 1996.

[80] D. Loguinov and H. Radha. Retransmission Schemes for Streaming Internet Multimedia: Evaluation Model and Performance Analysis. *ACM Computer Communication Review*, 32(2):70–83, April 2002.

[81] S. Low, F. Paganini, J. Wang, S. Adlakha, and J. Doyle. Dynamics of TCP/RED and a Scalable Control. In *Proc. IEEE INFOCOM '02*, New York, NY, June 2002.

[82] M. Luby, V. Goyal, S. Skaria, and G. Horn. Wave and Equation Based Rate Control Using Multicast Round Trip Time. In *Proc. ACM SIGCOMM '02*, Pittsburgh, PA, September 2002.

[83] A. Mahanti. Web Proxy Workload Characterisation and Modelling. Master's thesis, Department of Computer Science, University of Saskatchewan, September 1999.

[84] A. Mahanti, D. Eager, M. Vernon, and D. Sundaram-Stukel. Scalable On-Demand Media Streaming with Packet Loss Recovery. *IEEE/ACM Trans. on Networking*, 11(2):195–209, April 2003. An earlier version of this paper appeared in Proc. ACM SIGCOMM '01.

[85] A. Mahanti, C. Williamson, and D. Eager. Traffic Analysis of a Web Proxy Caching Hierarchy. *IEEE Network*, 14(3):187–203, May/June 2000.

[86] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgement Options. RFC 2018, April 1996.

[87] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The Macroscopic Behaviour of the TCP Congestion Avoidance Algorithm. *ACM Computer Communication Review*, 27(3):67–82, July 1997.

[88] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven Layered Multicast. In *Proc. ACM SIGCOMM '96*, Stanford, CA, September 1996.

[89] A. Mena and H. Heidemann. An Emperical Study of Real Audio Trace. In *Proc. IEEE INFOCOM '00*, Tel Aviv, Israel, March 2000.

[90] J. Mo, R. La, V. Anantharam, and J. Walrand. Analysis and Comparison of TCP Reno and Vegas. In *Proc. IEEE INFOCOM '99*, New York, NY, March 1999.

[91] S. Moon, P. Skelly, and D. Towsley. Estimation and Removal of Clock Skews from Network Delay Measurements. In *Proc. IEEE INFOCOM '99*, New York, NY, March 1999.

[92] J. Nonnenmacher, E. Biersack, and D. Towsley. Parity-Based Loss Recovery for Reliable Multicast Transmission. *IEEE/ACM Trans. on Networking*, 6(4):349–361, August 1998.

[93] J. Padhye, V. Firioiu, D. Towsley, and J. Kurose. Modelling TCP Throughput: A Simple Model and its Empirical Validation. In *Proc. ACM SIGCOMM '98*, Vancouver, Canada, September 1998.

[94] J. Padhye, J. Kurose, D. Towsley, and R. Koodli. A Model Based TCP Friendly Rate Control Protocol. In *Proc. NOSSDAV '99*, Basking Ridge, NJ, June 1999.

[95] J. Paris, S. Carter, and D. Long. Efficient Broadcasting Protocols for Video-on-Demand. In *Proc. MASCOTS '98*, Montreal, Canada, July 1998.

[96] J. Paris, S. Carter, and D. Long. A Hybrid Broadcasting Protocol for Video-on-Demand. In *Proc. MMCN'99*, San Jose, CA, January 1999.

[97] V. Paxon. On Calibrating Measurements of Packet Transit Times. In *Proc. ACM SIGMETRICS '98*, Madison, WI, June 1998.

[98] V. Paxson. End-to-End Internet Packet Dynamics. *IEEE/ACM Trans. on Networking*, 7(3):277–292, June 1999.

[99] V. Paxson and S. Floyd. Difficulties in Simulating the Internet. *IEEE/ACM Trans. on Networking*, 9(4):392–403, August 2001.

[100] C. Perkins, O. Hodson, and V. Hardman. A Survey of Packet Loss Recovery Techniques for Streaming Audio. *IEEE Network*, 12(5):40–48, September/October 1998.

[101] T. S. Perry. The Trials and Travails of Interactive TV. *IEEE Spectrum*, 33(4):22–28, April 1996.

[102] J. Ping, J. Kurose, and D. Towsley. Activating and Deactivating Repair Servers in Active Multicast Trees. In *Proc. IWDC '01*, Taormina, Italy, September 2001.

[103] NS Project. The Network Simulator: ns-2 (Web site). http://www.isi.edu/nsnam/ns.

[104] R. Rejaie, M. Handley, and D. Estrin. Quality Adaptation for Congestion Controlled Video Playback over the Internet. In *Proc. ACM SIGCOMM '99*, Cambridge, MA, September 1999.

[105] R. Rejaie, M. Handley, and D. Estrin. RAP: An End-to-End Congestion Control Mechanism for Realtime Streams in the Internet. In *Proc. IEEE INFOCOM '99*, New York, NY, March 1999.

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

[107] I. Rhee, N. Ballaguru, and G. Rouskas. MTCP:Scalable TCP-like Congestion Control for Reliable Multicast. In *Proc. IEEE INFOCOM '99*, New York, NY, March 1999.

[108] L. Rizzo. Effective Erasure Codes for Reliable Computer Communication Protocols. *ACM Computer Communication Review*, 27(2):24–36, April 1997.

[109] L. Rizzo. pgmcc: a TCP-friendly Single-Rate Multicast Congestion Control Scheme. In *Proc. ACM SIGCOMM '00*, Stockholm, Sweden, September 2000.

[110] L. Rizzo and L. Vicisano. A Reliable Multicast Data Distribution Protocol Based on Software FEC Techniques. In *Proc. HPCS '97*, Chalkidiki, Greece, June 1997.

[111] L. Sahasrabuddhe and B. Mukherjee. Multicast Routing Algorithms and Protocols: A Tutorial. *IEEE Network*, 14(1):90–102, January/February 2000.

[112] B. Samois and M. Vernon. Modeling the Throughput of TCP Vegas. In *Proc. ACM SIGMETRICS '03*, San Diego, CA, June 2003.

[113] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications, January 1996. RFC 1889.

[114] H. Schulzrinne, A. Rao, and R. Lanphier. Real Time Streaming Protocol (RTSP), April 1998. RFC 2326.

[115] S. Sen, L. Gao, J. Rexford, and D. Towsley. Optimal Patching Schemes for Efficient Multimedia Streaming. In *Proc. NOSSDAV '99*, Basking Ridge, NJ, June 1999.

[116] S. Sen, J. Rexford, and D. Towsley. Proxy Prefix Caching for Multimedia Streams. In *Proc. IEEE INFOCOM '99*, New York, NY, March 1999.

[117] N. Shacham. Multipoint Communication by Hierarchically Encoded Data. In *Proc. IEEE INFOCOM '92*, Florence, Italy, May 1992.

[118] D. Sisalem and H. Schulzrinne. The Loss-Delay Based Adjustment Algorithm: A TCP-Friendly Adaptation Scheme. In *Proc. NOSSDAV '98*, Cambridge, UK, July 1998.

[119] D. Sisalem and A. Wolisz. MLDA: A TCP-friendly Congestion Control Framework for Heterogeneous Multicast Environments. In *Proc.* 8^{th} *Int'l. Workshop on QoS*, Pittsburgh, PA, June 2000.

[120] W. Stevens. *TCP/IP Illustrated*, volume 1. Addison-Wesley Longman Inc., 1994.

[121] D. Sundaram-Stukel, M. Vernon, J. Zahorjan, and D. Eager. SWORD Prototype Document. Internal Technical Report.

[122] W. Tan and A. Zakhor. Multicast Transmission of Scalable Video using Receiver-driven Hierarchical FEC. In *Packet Video Workshop*, New York, NY, April 1999.

[123] W. Tan and A. Zakhor. Real-time Internet Video Using Error Resilient Scalable Compression and TCP-friendly Transport Protocol. *IEEE Trans. Multimedia*, 1(2):172–186, June 1999.

[124] A. Tsiolis and M. Vernon. Group-Guaranteed Channel Capacity in Multimedia Storage Servers. In *Proc. ACM SIGMETRICS '97*, Seattle, WA, June 1997.

[125] T. Turletti, S. Parisis, and J. Bolot. Experiments with a Layered Transmission Scheme over the Internet. INRIA Tech. Report 3296, November 1997.

[126] L. Vicisano, L. Rizzo, and J. Crowcroft. TCP-like Congestion Control for Layered Video Multicast Data Transfer. In *Proc. IEEE INFOCOM '98*, San Francisco, CA, April 1998.

[127] S. Viswanathan and T. Imielinski. Metropolitan Area Video-on-Demand Service using Pyramid Broadcasting. *Multimedia Systems*, 4(4):197–208, August 1996.

[128] D. Waitzman, C. Partridge, and S. Deering. Distance Vector Multicast Routing Protocol (DVMRP). RFC 1075, November 1988.

[129] H. Wang and M. Schwartz. Achieving Bounded Fairness for Multicast and TCP Traffic in the Internet. In *Proc. ACM SIGCOMM '98*, Vancouver, Canada, September 1998.

[130] Y. Wang, Z. Zhang, D. Du, and D. Su. A Network Concious Approach to End-to-End Video Delivery Over Wide Area Networks Using Proxy Servers. In *Proc. IEEE INFOCOM '98*, San Francisco, CA, Mar. 1998.

[131] J. Widmer, R. Dende, and M. Mauve. A Survey of TCP-friendly Congestion Control. *IEEE Network*, 15(3):28–37, May 2001.

[132] J. Widmer and M. Handley. Extending Equation-Based Congestion Control to Multicast Applications. In *Proc. ACM SIGCOMM '01*, San Diego, CA, August 2001.

[133] J. Wong. Broadcast Delivery. *Proc. IEEE*, 76(12):1566–1577, December 1988.

[134] X. Xu, A. Myers, H. Zhang, and R. Yavatkar. Resilient Multicast Support for Continuous-Media Applications. In *Proc. NOSSDAV '97*, St. Louis, MO, May 1997.

[135] R. Yavatkar, J. Griffoen, and M. Sudan. A Reliable Dissemination Protocol for Interactive Collaborative Applications. In *Proc. ACM MULTIMEDIA '95*, pages 333–344, San Francisco, CA, 1995.

Appendix A

TCP Reno Throughput Models

Models have been proposed that predict the steady state throughput of a long duration TCP Reno flow as a function of the average RTT (R) and the loss event rate (p) [87, 93, 57]. This section revisits a simple formulation of the throughput function that can be derived under the assumption that all loss indications are via triple duplicate acknowledgments, loss events have a fixed probability and are mutually independent, and RTT and loss event rate are independent of the sending rate [87]. Figure A.1 illustrates the evolution of the congestion window through several increase/decrease cycles when all loss indications are via triple duplicate acknowledgments. The period between two packet loss events is called an "epoch", and the throughput of an epoch can be calculated as the ratio of the expected number of packets sent in an epoch (P_e) to the expected duration of an epoch (D_e). The number of packets transmitted in an epoch is $\frac{3W^2}{8}$, and the duration of an epoch is $\frac{W}{2}R$, assuming the window size W increases by one every RTT in the congestion avoidance phase. Using the fact that a loss event rate of p implies $1/p$ packets are transmitted in an epoch, a relationship between p and W can be obtained as follows: $W = \sqrt{\frac{8}{3p}}$. The estimated throughput, in packets per second, is given by [87]:

$$\Lambda^{reno}_{loss\ model} = \frac{P_e}{D_e} = \frac{3W}{4R} = \sqrt{\frac{3}{2p}}\frac{1}{R}. \tag{A.1}$$

The model described above will be referred to as the "loss model" in the rest of this section. An alternative expression for TCP Reno throughput can be derived by considering the the time between congestion window reductions (or equivalently, the duration of an epoch), instead of the probability of first packet loss. That is,

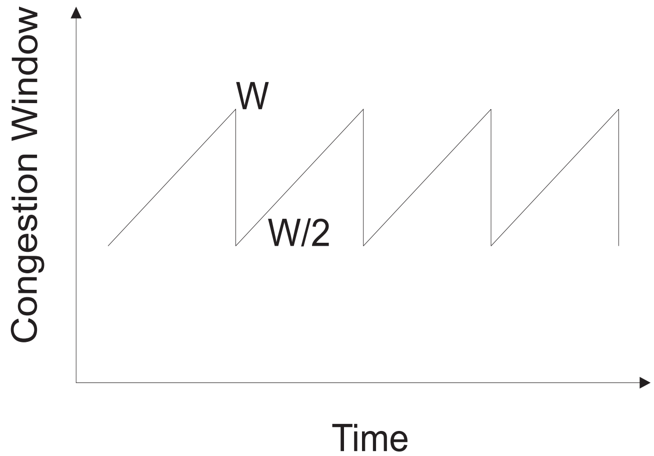


Figure A.1: TCP Reno Window Evolution Assuming all Loss Indications are by Triple Duplicate Acknowledgments

noting that $D_e = \frac{W}{2}R$, the equivalent expression for throughput is:

$$\Lambda^{reno}_{time\ model} = \frac{3D_e}{2R^2}. \tag{A.2}$$

This model will be referred to as the "time model" in the remainder of this discussion. It is interesting to note that the "loss model" indicates that throughput varies linearly with the inverse of RTT, while the "time model" indicates that throughput varies as the inverse of the square of the RTT. The next section explores this issue.

A.1 Impact of Different RTT on Throughput

The impact of RTT on the throughput observed by a TCP Reno flow has been much debated. For example, under assumption of synchronized flows and droptail queue management, the ratio of the TCP throughputs for two flows is proportional to the inverse of the square of their RTTs [72, 14]. Others have reported that the ratio of the TCP throughputs is proportional to the inverse of their RTTs [52]. Experiments reported in this section confirm both observations. That is, depending on the topology and characteristics of background traffic, it is shown that either observation can be obtained.

The first experiment uses a dumbbell topology with a single bottleneck link

Table A.1: Impact of RTT on Throughput

RTT for Group 1 (R_1)	RTT for Group 2 (R_2)	Throughput for Group 1 (X_1)	Throughput for Group 2 (X_1)	R_2/R_1	X_1/X_2
0.066	0.124	136.81	48.48	1.87	2.82
0.068	0.166	132.15	50.00	2.42	2.64
0.067	0.218	152.87	37.21	3.23	4.11
0.068	0.336	163.88	19.72	4.95	8.31

of capacity 2 Mbps with a droptail queue of 25 packets. Two FTP (using TCP Newreno) flows, each having a different RTT, share the bottleneck link. To perturb the bottleneck link, an exponential ON/OFF traffic at rate 500 Kbps is used. Results from this simulation showed that the ratio of throughputs is proportional to the square of the inverse of the ratio of RTTs.

The next experiment extends the above simulation scenario to consider a slightly richer mix of traffic. The setup considers 20 FTP flows sharing a bottleneck link with bandwidth 10 Mbps and a droptail queue of 50 packets. These 20 flows can be divided into two groups, each with equal numbers of flows, such that one group (Group 1) has a shorter propagation delay than the other (Group 2). To remove the possibility of synchronization of the window evolutions, the following precautions are taken: 1) 4 FTP flows are employed in the reverse path; 2) an exponential ON/OFF UDP traffic at rate 500 Kb traverses the bottleneck link; and 3) all flows start at slightly different times. The results shown in Table A.1 indicate that the ratio of the average throughput of each group is between the inverse of the ratio of average RTT and the inverse of the ratio of square of average RTT of the groups.

The third experiment considers a dumbbell topology with a bottleneck link of capacity 15 Mbps with a 100 packet droptail queue. Background traffic consists of 10 FTP Newreno flows and 10 HTTP ON/OFF flows. The round-trip propagation delays of the background flows is uniformly distributed in $[50, 410]$ milliseconds. Two foreground FTP Newreno flows are considered. The round trip propagation delay of one flow is fixed at 50 milliseconds, while the round-trip propagation delay of the other flow is varied. Results from 5 simulation runs are shown in Figure A.2. The

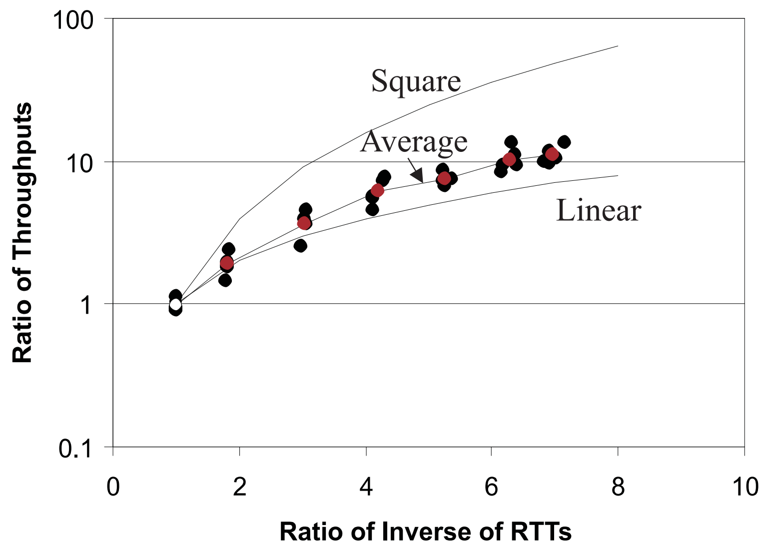


Figure A.2: Illustrating the RTT Bias of TCP Reno

results confirm the observations made from the previous experiment.

A.2 Understanding the Control Equations

Ideally, a "TCP-friendly" UDP flow and a competing FTP flow (using TCP Reno) should share the bottleneck resources equally. This section attempts to understand how the throughput equations discussed above can guide a UDP flow towards its "TCP-friendly" bandwidth share. The two main issues considered here are:

- Under what circumstances are the loss rates experienced by the UDP flow and the FTP flow roughly similar? There is evidence in the literature that points out that the loss rates seen by the two flows often differ, when droptail queuing is employed [13].

- Given a UDP flow at a certain rate, do the throughput equations considered in the preceding section generate a signal that can be used to steer the UDP flow to reach an equilibrium condition with a foreground FTP flow (i.e., both the UDP and FTP flow have approximately similar throughput shares)?

The above questions are answered by conducting *ns* simulations. The simulations consider a dumbbell topology with a bottleneck link of capacity 15 Mbps and a

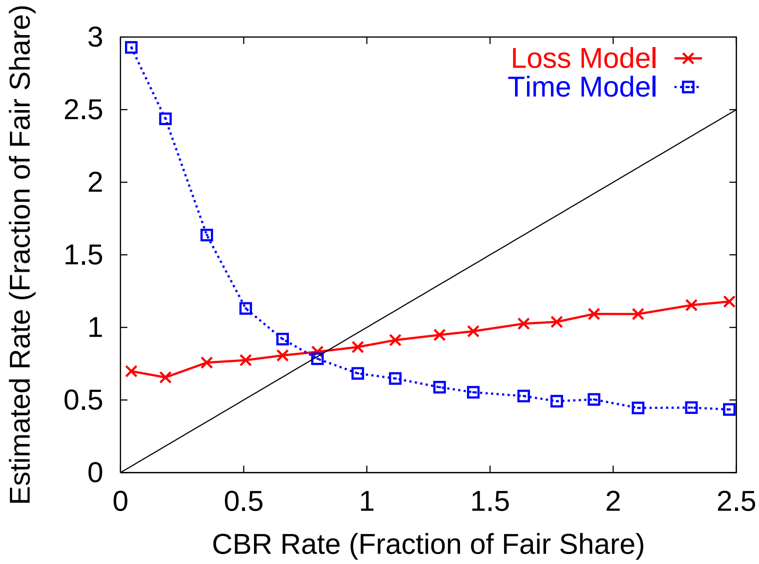


Figure A.3: Relative Aggressiveness of the TCP Reno Throughput Models

buffer of 100 packets. The queues employ the droptail queuing discipline. The foreground traffic consists of a UDP flow and a FTP flow, both having the same round trip delays. Background traffic consists of a fix of 15 FTP flows and 15 HTTP sessions. The background flows have round-trip propagation delays uniformly distributed between $[20, 460]$ milliseconds.

A number of simulation runs are carried out using the above described setup. Each run considers a UDP flow at a certain constant bit rate. For any simulation run, the (ideal) "TCP-friendly" bandwidth share is deemed to be equal to the throughput achieved by the competing foreground FTP traffic. Also, for each run, the number of loss events witnessed by the UDP flow is recorded and used to determine its loss event rate. Similarly, the average time between loss events can also be determined for the UDP flow.[1] The fair share throughput estimate, as obtained by application of the TCP throughput models, can then be determined.

Figure A.3 presents results from a simulation where the foreground flows have a round-trip propagation delay of 150 milliseconds. In the figure, the x-axis corresponds to the current UDP flow rate, while the y-axis represents the estimate

[1] All losses within two round trip times are considered part of the same loss episode. The average round trip time experienced by the foreground FTP flow is used as the round trip time value for the UDP flow.

obtained by application of the throughput formula (both, normalized with respect to the current "TCP-friendly" bandwidth share). In general, both models provide increase/decrease signals that can be used to guide a UDP flow towards the equilibrium point (at which the UDP flow and the FTP flow have similar throughputs). For example, when the UDP flow's rate is less than the rate of the FTP flow, the models indicate that the rate of the UDP flow should be increased. In the figure, this corresponds to all points that fall below the straight line. Also note that the "time model" provides much more aggressive increase/decrease signals than the "loss model", when the UDP flow rate is higher/lower than the "fair share" bandwidth. This also indicates that the loss event rate (and also the time between loss events) seen by the UDP flow can differ from that witnessed by the FTP flow.

A detailed study of these alternative models is beyond the scope of this research, and is a fruitful direction for future work.