Scalable and Interactive Multimedia Streaming over the Internet

by

Md Humayun Kabir
B.Sc., Bangladesh University of Engineering and Technology, Dhaka, 1993
M.Sc., Bangladesh University of Engineering and Technology, Dhaka, 1998

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in the Department of Computer Science

© Md Humayun Kabir, 2005
University of Victoria

All rights reserved. This dissertation may not be reproduced in whole or in part, by photocopy or other means, without the permission of the author.

Supervisors: Dr. E. G. Manning and Dr. G. C. Shoja

ABSTRACT

Streaming audio/video contents over the Internet requires large network bandwidth and timely delivery and playback of the media data. However, large network latency and jitter cause long start-up delay and frequent unwanted pauses in the playback, respectively. An entire audio/video media file cannot be cached due to intellectual property right concerns of the content owners, security reasons, and also due to its large size. This makes a streaming service hard to scale using conventional proxy caches. Media file compression using variable-bit-rate (VBR) encoding is often preferred in order to get constant quality compressed videos. VBR-encoding produces traffic burst, which not only wastes bandwidth but also inserts hiccup in the media playback. The standard frame sequence of a compressed video stream is only suitable for normal playback. Its inter-frame dependency makes it difficult to play it in interactive playback modes, such as fast forward/backward, jump and play backward. Moreover, different interactive clients generally ask for different playback sequences. Hence, they cannot be served together using a common server stream. Therefore, as the frequency of interaction increases, an ordinary scalable streaming service transforms into a non-scalable service.

In this thesis, we present a new proxy based constant-bit-rate (CBR) streaming scheme that allows a server to transmit a VBR-encoded video at a fixed rate, close to its mean encoding bit-rate, and deals with the network latency and jitter issues efficiently without caching an entire media file at the proxy. We use a prefix buffer at the proxy to cache the prefixes of popular videos in order to minimize the start-up delay and to enable near mean bit rate streaming. We present a new proxy based scalable streaming scheme that uses our CBR streaming scheme. We use the smoothing buffer at the proxy not only to eliminate jitter and traffic burst effects but also to enable many clients to share the same

server stream. We also present a new interactive and scalable streaming scheme, which divides a video stream into several logical segments and provides segment-by-segment interactive playback options to the clients. We use hybrid temporal-data-partition scalable encoding to create a suitable playback sequence for the interactive playback modes. Experimental result shows that our streaming scheme remains fully scalable even when all the clients are highly interactive. As further improvements, we present a new on demand and user driven segmentation and proxy buffer provisioning (prefix caching) mechanism for our interactive and scalable streaming scheme in order to avoid buffer over provisioning at the proxy as well as to avoid the use of complex video segmentation algorithms. We also present a new collaborative-proxy-peering system in order to get better resource utilization and performance from a set of proxies that are used to stream a video.

Mathematical expressions to compute the precise sizes of the prefix and the smoothing buffers as well as the precise amounts of bandwidth requirements have been developed. All our streaming schemes have been analyzed and the results of Java simulation programs have shown their effectiveness.

Supervisors: Dr. Eric G. Manning and Dr. G. C. Shoja, (Department of Computer Science)

Table of Contents

List of Figures

List of Symbols

b	Mean bit rate of a video
b_i	Mean bit rate of video i
B	Total amount of buffer required for a video
$B_{isp,non-collaborative}$	Total amount of buffer required for a video for non-collaborative proxies
$B_{isp,collaborative}$	Total amount of buffer required for a video for collaborative proxies
c_{st}	Cost of transmitting one bit of video data from the server to the proxy
c_{pt}	Cost of transmitting one bit of video data from the proxy to the client
c_m	Cost of memory to buffer one bit of video data
C_i	Total cost (transmission plus buffering) for video i
$C_{i,average}$	Average streaming cost per client for video i
$d_{s,p}^{\max}$	Maximum server-to-proxy network latency
$d_{s,p}^{\min}$	Minimum server-to-proxy network latency
$d_{p,c}$	Proxy-to-client network latency
d_s	Playback start-up delay
$\Delta_{s,p}^{\max}$	Maximum server-to-proxy jitter

D_{burst}	Burst-work-ahead data of a video in bits
$D_{l,burst}$	Burst-work-ahead data of segment S_l in bits
D_{cw}	Size of the client-work-ahead data in bits
$E(S)$	Average size of the segments of a video in seconds
f_{rate}	Frame rate of a video, frames/second
f_{mean}	Average frame size of a video
f_j	Size of j^{th} frame of a video
$f_{j,l,b}$	Size of the j^{th} frame of the base sub-stream of segment S_l
$f_{j,l,e}$	Size of the j^{th} frame of the enhancement sub-stream of segment S_l
h	Number of bits in an identifier generated by a consistent hash function
I	Number of interactive playback requests in a single video duration
k	Key of a data that can be placed on a Chord node
L	Duration of a video in seconds
L_i	Duration of video i in seconds
L_b	Average size of the base sub-stream of a video in seconds
λ	Request arrival rate of a video, requests/second
λ_i	Request arrival rate of video I, requests/second

M	Size of the whole video in bits
m	Number of frames in the minimum segment length
n	Number of total frames in a video
n_s	Number of segments in a video
n_l	Average number of frames in each segment
N	A node identifier in the Chord ring
$N_{play,smooth}$	Average number of smoothing buffers required for normal playback of a video
$N_{\text{int},smooth}$	Average number of smoothing buffers required for interactive playback
P_{play}	Probability of normal playback of a video
$P_n(s)$	Probability that s video segments are requested for normal playback
$P_{\text{int}}(s)$	Probability that s video segments are required for interactive playback
$P_{seg,s}$	Probability that s segments have been created in a single video duration
$P_{n\_smooth,i}$	Probability that i smoothing buffers have been allocated for a segment for normal playback
$P_{\text{int}\_smooth,i}$	Probability that i smoothing buffers have been allocated for a segment for interactive playback
Q_q	Maximum number of concurrent clients that a single proxy can serve
r	Transmission rate ratio to the mean bit rate of a video

$r_{s,p}$	Server-proxy transmission rate-ratio
$r_{p,c}$	Proxy-client transmission rate-ratio
R	Set of all real numbers
S_l	Logical segment l of a video
$S_{l,e}$	Enhancement sub-stream of segment S_l
$S_{l,b}$	Base sub-stream of segment S_l
$S_{\min,length}$	Minimum-segment-length of a video in seconds
$S_{\max,number}$	Maximum possible number of segments of a video
$S_{avg,number}$	Average number of segments created in a single video duration
S_{time}	Average segmentation time necessary to reach the segmentation saturation point
$S_{time,non-collborative}$	Average segmentation time for non-collaborative proxies
$S_{time,collborative}$	Average segmentation time for collaborative proxies
$S_{avg,length}$	Average segment length in seconds
S_{slot}	Number of sharing slots in a segment
$S_{n\_s\_buffer}$	Average number of smoothing buffers for normal playback of a segment
$S_{\text{int}\_s\_buffer}$	Average number of smoothing buffers for interactive playback for a segment

T_{prefix}	Size of the prefix buffer of a video in seconds
T_{burst}	Burst-work-ahead time
$T_{playback}$	Size of the playback buffer at the client in seconds
$T_{smoothing}$	Size of the smoothing buffer for a video in seconds
$T_{workahead}$	Size of the work-ahead smoothing buffer for a video in seconds
$T_{sharing}$	Size of the sharing-smoothing-buffer of a video in seconds
$T_{i,sharing}$	Size of the sharing-smoothing-buffer of video i in seconds
$T_{l,prefix}$	Size of the prefix of segment S_l in seconds
$T_{l,smoothing}$	Size of the smoothing buffer for segment S_l in seconds
$T_{average,smoothing}$	Average size of the segment suffixes/smoothing buffers in seconds
T_{pa}	Pause duration
U	Set of all positive integers
x	Fast forward/backward speed up factor
X	GOP size of a video
χ	Average server-proxy bandwidth required for a video in bps
$\chi_{isp,non-collaborative}$	Average server-proxy bandwidth required for non-collaborative proxies
$\chi_{isp,collaborative}$	Average server-proxy bandwidth required for collaborative proxies

Y	Interval between two consecutive P-frames in a GOP
w	Number of low frequency DCT coefficient in the base sub-stream

List of Acronyms

ADSL	Asymmetric DSL
ARPANET	Advance Research Projects Agency Network
AS	Autonomous System
ATM	Asynchronous Transfer Mode
BE	Best Effort
BGP	Border Gateway Protocol
CBR	Constant-bit-rate
CCP	Chord Coordinator Program
CDN	Content Distribution Network
CGS	Coarse Granular Scalable
CMTS	Cable Modem Terminal Server
CNAME	Canonical Name
CO	Central Office
CPPS	Collaborative Proxy-peering System
CRC	Cyclic Redundancy Code
DC	Direct Current/ Direct Component
DCT	Discrete Cosine Transformation
DiffServ	Differentiated Service
DOCSIS	Data Over Cable Service Interface Specification
DPCM	Differential Pulse Code Modulation

DS	Descriptor Schema
DSCP	Differential Service Code Point
DSL	Digital Subscriber Line
DSLAM	DSL Access Multiplexer
DTS	Decode Time Stamp
EF	Expedited Forwarding
ERA	Expanding Ring Advertisement
ES	Elementary Stream
ESCR	Elementary Stream Clock Reference
FB	Fast Backward
FF	Fast Forward
FGS	Fine Granular Scalable
FTTH	Fiber to the Home
GigE	Gigabit-Ethernet
GMPLS	Generalized MPLS
GOP	Group of Pictures
HDSL	High data rate DSL
HTTP	Hypertext Transfer Protocol
ID	Identifier
IDCT	Inverse-DCT
IDSL	ISDN DSL
IntServ	Integrated Service
IP	Internet Protocol

ISP	Internet Service Providers
IVoD	Internet Video-on-Demand
JB	Jump Backward
JF	Jump Forward
LSR	Label Switched Router
MPEG	Motion Picture Experts Group
MPLS	Multi-protocol Label Switching
MTU	Maximum Transmission Unit
OC	Optical Carrier
ODN	Optical Distribution Network
OLT	Optical Line Terminator
ONU	Optical Network Unit
OOO	Optical-Optical-Optical
OSPF	Open Shortest Path First
PA	Pause
PB	Play Backward
PCM	Pulse Code Modulation
PDA	Personal Digital Assistant
PES.	Packetized Elementary Stream
PON	Passive Optical Network
POTS	Plain Old Telephone Service
PSTN	Public Switched Telephone Network
PTS	Presentation Time Stamp

QAM	Quadrature Amplitude Modulation
QoS	Quality of Service
QPSK	Quadrature Phase Shift Keying
RED	Random Early Detection
RFC	Request for Comments
RLE	Run Length Encoding
RR	Receiver Report
RSVP	Resource Reservation Protocol
RTCP	RTP Control Protocol
RTP	Real-time Transport Protocol
RTSP	Real Time Streaming Protocol
SAM	Split and Merge
SB	Slow Backward
SDES	Source Description
SDP	Session Description Protocol
SDSL	Single-Line DSL
SF	Slow Forward
SHA	Secured Hash
SI	Switching I
SNR	Signal-to-Noise-Ratio
SP	Switching P
SR	Sender Report
SSRC	Synchronization Source Identifier

TCP	Transmission Control Protocol
TOS	Type of Service
UDP	User Datagram Protocol
URI	Universal Resource Identifier
URL	Uniform Resource Locator
USB	Universal Serial Bus
VBR	Variable-bit-rate
VCR	Video Cassette Recorder
VDSL	Very high data rate DSL
VoIP	Voice-over-IP
WDM	Wavelength Division Multiplexing
WiFi	Wireless Fidelity
WiMAX	Worldwide Interoperability for Microwave Access

Acknowledgement

I would like to express my deep appreciation to my supervisors Dr. Eric G. Manning and Dr. Gholamali C. Shoja for their continuous support and guidance. Their thoughtful suggestions were vital to shape my ideas into this dissertation. I would also like to thank members of my dissertation committee Dr. John Ellis and Dr. Fayez Gebali for their valuable time and suggestions.

I like to thank all the members at the PANDA research lab. They were always great source of help. Feedbacks after the PANDA seminars were also valuable and enlightening. I like to give special thank to my long-time officemate Steven Shelford. Our regular discussion helped us to clear lots of computer networking related concepts. Steve was always cooperative. Discussions with Dr. Ganti also helped me to enrich my computer networking knowledge. I like to thank Feng and Steve for helping me in submitting my request for oral examination form while I was out of Victoria. Thanks to Doug for helping me in finding cool things from Vancouver.

The unending love and inspiration of my family members gave me strong support for this research. I like to express my gratefulness to my family members, which includes my wife, daughter, parents, sisters, and brother.

Bangladeshi friends in Victoria were my endless source of joy. My special thank to Md. Mostofa Akbar for helping me a lot at the beginning of my life in Victoria.

To my mom Jahanara Begum

and

to my wife and daughter

Maria Wahid Chowdhury and Tanha Kabir

1. Introduction

Video streaming applications over the Internet, such as videoconferencing, video-on-demand, and videophone, have tremendous business potential. However, the problems present in video streaming over the Internet hinder businesses from benefiting from its potentials. The main problems in video streaming are as follows:

i. It needs quality-of-service (QoS) guarantees with respect to bandwidth, delay, jitter, and data loss from the Internet,

ii. It is bandwidth, memory and computation intensive,

iii. It faces traffic bursts often produced by the compressed videos,

iv. It is inherently non-scalable, and

v. It needs to be interactive.

Solving these problems is an interesting research area in Computer Science, which involves Communication Networks, Distributed Computing, and Media Compression. We address some of above issues in our research. We present a proxy based video streaming scheme over the Internet, which is both bandwidth saving and scalable. Our proposed scheme enables the Internet Service Providers (ISPs), such as the telephone companies or the cable companies, to take full benefit from the business potentials of streaming applications, particularly from video-on-demand applications. It absorbs network delay, jitter, and the traffic bursts that are produced by the compressed videos. Our solution provides interactive playback modes of the videos to the users, and allows on-demand video segmentation and buffer provisioning at the proxy for interactive playback. It enables multiple proxies in an ISP to collaborate with each other in order to achieve better bandwidth and buffer utilization.

1.1. Motivation

The Internet has emerged as a pervasive communication medium, which is used as a ubiquitous technology to transmit, as well as to disseminate, content from anywhere to anywhere in the world. Video streaming over the Internet also has potentially huge commercial value. In order to tap this value, it is necessary to solve the problems that hinder its commercialization. Protocols, such as Real-time Transport Protocol (RTP) [1] and Real Time Streaming Protocol (RTSP) [2], can be used to stream video over the Internet. QoS guarantees on the Internet have become possible through some mechanisms and architectures, such as Integrated Service (IntServ) [4] and Differentiated Service (DiffServ) [5], and use of Multi-protocol Label Switching (MPLS) [7]. The Lambda (λ)-switched Internet [6] promises to reduce both delay and jitter and also to bound them. Deployment of proxy caches [8] and Content Distribution Networks (CDNs) [9] in the Internet can minimize the delay and the jitter perceived by clients, who may be geographically distributed around the globe, and reduce the load on the origin server. Video encoding methods, such as H.261 [10], MPEG-2 [11][12][13], and MPEG-4 [14][15], compress large video files. Video file compression reduces both the memory and the bandwidth requirements to a large extent. These recent developments in media encoding and in the Internet are major breakthroughs for video streaming over the Internet, which entice both the research and the business communities, including us, to solve the remaining problems.

A side effect of the video encoding techniques listed above is production of traffic bursts. Due to traffic bursts, peak bandwidth requirements remain high, and thus the overall streaming bandwidth requirement remains high. Compressed video files are still large, e.g., a 2-hour MPEG-2 video is around 5GB in size. Due to its large bandwidth, memory, and computational requirements as well as its great sensitivity to delay, jitter, and loss, a streaming service inherently remains non-scalable. Ordinary proxy caches and CDN surrogate servers, which have been used to ease delay, jitter and server load problems for ordinary web objects, such as html files and images, cannot be used for video streaming.

A compressed frame sequence of a video is only suitable for normal playback. Different interactive playback modes need different playback sequences that are not generated by simple video encoding. Different interactive clients also may ask for different interactive playback modes, i.e., different playback sequences. It is hard to serve these clients concurrently using a common playback sequence. For this reason, an interactive playback requirement increases the difficulty of scaling a streaming service.

A commercially viable video streaming service must be scalable, frugal in bandwidth and memory required, and should provide several interactive playback modes, as customers who have used VCRs and DVD players expect to have these. Designing such a video streaming service needs comprehensive understanding and research about the problems and related issues.

1.2. Problem Definition and Previous Work

A video streaming system over the Internet consists of a video server or a video-server farm and a large number of clients, assumed to be geographically distributed all around the world, as shown in Figure 1.1.

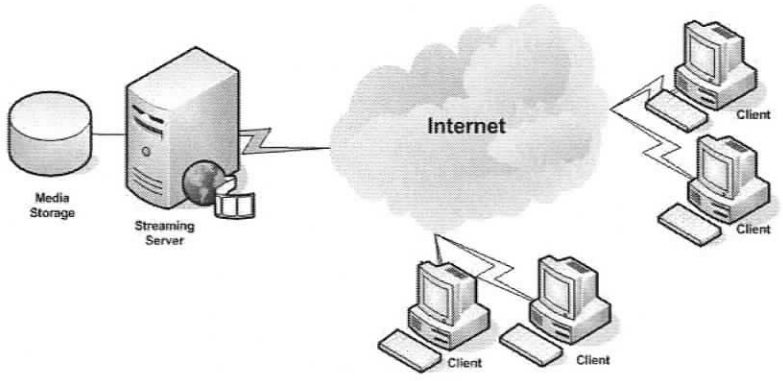

Figure 1.1: Streaming System over the Internet

Clients connect to the server through the Internet. The server maintains the compressed videos in its media storage and fetches them from the storage when requested. Clients

request video streams from the server using a real-time streaming protocol (RTSP)[2]. The server transmits the video data to the clients using a real-time transport protocol, such as RTP [1]. After receiving the video data, a client decodes and plays it in real time. A client session generally continues for a long time, in the order of an hour, until the end of a video.

The present Internet is mostly only unicast enabled –multicast is not commonly available. For this reason, each client needs a separate RTP connection i.e., separate server memory and network bandwidth, in order to retrieve the video data from the server. The memory and network bandwidth of a server are limited. A scalable video streaming system needs to be able to transmit videos to a very large number of clients using the available memory and network bandwidth. The distance between the server and the client often results in large network latency and jitter, causing large start-up delay. Network jitter also causes hiccups in the playback. A good streaming system needs to overcome latency and jitter problems so that the clients can have quick start-up and hiccup free playback.

A video file is compressed using either constant-bit-rate (CBR) encoding or variable-bit-rate (VBR) encoding [16]. CBR-encoding generates a constant bit rate stream, which is easy to transmit at a fixed bit rate equal to its encoding bit rate. However, it reduces the degrees of freedom for the encoding task and reduces the quality of those video intervals, which are full of action. For these reasons, VBR-encoding, which produces constant quality videos, is preferred to CBR-encoding. VBR-encoding, however, creates frequent traffic bursts. Frequent traffic bursts may insert frequent hiccups in the playback and need large peak bandwidth. Frequent hiccups in the playback cause client dissatisfaction, i.e., the loss of business. The large peak bandwidth requirement in turn limits the number of clients that can be served by the server, causing further loss of business. A scalable streaming system has to smooth out the traffic burst in order to remove the hiccup from the playback as well as to reduce the peak bandwidth requirement, so that the server can serve a large number of clients with the available bandwidth.

It is not feasible to cache the entire file of a streaming media at a streaming proxy, for two reasons. First, streaming media files are typically very large. Secondly, the owners of the streaming media content do not want the proxy or the client to cache the whole media file, for business and security reasons. These issues make it hard to make a streaming service scalable using conventional proxy caches. A scalable streaming system needs to use a special streaming proxy in order to hide network latency and jitter without caching the whole media file at the proxy. A streaming proxy also needs to smooth out the traffic bursts of the compressed videos.

Sen et al. [17] use a special proxy with prefix and work-ahead smoothing buffers to mask network latency and jitter and to do online smoothing of VBR traffic bursts. However, they do online smoothing on the proxy-client path but not on the server-proxy path. Each client needs a separate smoothing buffer at the proxy as well as a separate stream from the server, i.e., this scheme is entirely non-scalable. Wang et al. [18] use a proxy with prefix buffers to mask network latency and to enable several clients to share the same server stream, in order to make the streaming service scalable. However, it can handle only CBR-encoded videos. They do not use any kind of work-ahead smoothing operation to handle network jitter problem and traffic burst problem of a VBR-encoded video. Boroczky et al. [19] propose a technique to statistically multiplex several VBR-encoded streams into a larger, less bursty stream, and hence one closer to a CBR stream in order to minimize the server bandwidth requirement. Theirs is a complex technique. It is also ineffective when a large number of clients request a popular video at the same time. There are several online smoothing techniques [20][21][22] that pre-fetch the prefix of a video in a buffer at the proxy or at the client and convert a VBR stream into several consecutive runs of different CBR streams, thus reducing the peak rate as well as the variability of data rates of VBR-encoded stored media. These smoothing techniques do not reduce the peak bit rate to the mean bit rate and do not eliminate the rate variability completely; rather they add server complexities.

A good streaming system should provide several interactive playback modes in order to provide convenient playback options to the clients. The commonly required interactive playback modes are

- Play Normal (Forward)
- Play Backward (PB)
- Pause (PA)
- Jump Forward (JF)
- Jump Backward (JB)
- Fast Forward (FF)
- Fast Backward (FB)
- Slow Forward (SF) and
- Slow Backward (SB).

Video encoders, such as H.261 [10], MPEG-2 [12][13], and MPEG-4 [14][15] generally create three types of compressed frames: I-frames, P-frames, and B-frames. A typical frame sequence of a compressed video stream is I-B-B-P-B-B-P-B-B-P-B-B-I-B-B-P-, which is only suitable for normal playback. Motion compensated forward predictive coding makes it difficult to play the above sequence in an interactive playback mode. A scalable streaming scheme generally requires sharing a single server stream among a group of clients. An interactive client, however, needs a unique playback sequence; hence, multiple interactive clients cannot be served by using a single server stream. Thus, as the number of interactive clients rises, an ordinary scalable streaming service tends to become less scalable. A good scalable streaming system should be able to generate different playback sequences for different playback modes and should remain scalable even when the number of interactive clients rises.

Lin et al. [23] proposed dual bit-streams: a forward encoded stream for forward playback and a special reverse encoded stream for reverse playback. Two more bit-streams consisting of all P-frames are used to compensate for drift during bit-stream switching. Tourapis et al. [24] used a separately encoded stream named Intelligent Interactive Stream (I^2 Stream). They have also proposed to use two more bit-streams to compensate for drift during bit-stream switching. Adding too many bit-streams requires more disk space at the server, which makes the schemes of both in [23] and [24] less useful.

Gao et al. [25] proposed to use scalable encoding in order to create a base sub-stream and an enhancement sub-stream; they use only the base sub-stream for fast forward playback. Their scheme works well with a simple streaming model, which has a streaming server and some clients. However, it does not work with a scalable streaming scheme, which may share the same server stream among many clients through one or more proxies. Their proposed layered base soft real-time scheduling algorithm supports only fast forward playback.

Chen et al. [26] proposed a straightforward implementation of fast backward playback. It transmits and plays only the I-frames in the reverse order. Transmitting only the I-frames at a higher speed requires more bandwidth, of course. Krunz et al. [27] have proposed to use a separately encoded low-resolution fast backward stream also for fast backward playback. Separately encoded bit-streams need large storage at the server. Feng et al. [28] proposed a buffer based scheme, where the most recently received frames are stored in a client buffer called the VCR-window. All the backward VCR capabilities, such as FB, JB and SB, are fully functional within the VCR-window. However, this scheme has several limitations, for example every client needs to maintain a large buffer or a VCR-window and it cannot support any forward operations.

Liao et al. [29] proposed a Split and Merge (SAM) protocol to minimize the problem of supporting VCR functionalities in a shared streaming model. SAM starts by serving clients in a group. When a client in some group initiates a VCR interaction, the protocol splits off the interactive client from its original group and temporarily assigns a new video stream called an interactive stream to that client. With a dedicated video stream the client can perform any VCR operation that it desires. As soon as the VCR operation terminates, SAM merges this client with a nearest running group. The problem with the SAM protocol is that it needs many dedicated interactive streams when many clients request VCR operations, i.e., it loses scalability as interaction increases. Also, different interactive playback modes need different types of playback sequences. SAM does not specify which playback sequence is being transmitted through the dedicated streams for which interactive playback mode.

Keeping the provision of different playback modes at the proxy often requires large buffer space. If the demand for interactive playback is not high, pre-provisioning large buffer space for interactive playback is simply a waste of resources. A good scalable streaming system should avoid this waste by provisioning resources on demand.

The bandwidth and the memory available at a proxy are also fixed. A single proxy in an ISP might not have enough bandwidth or memory to serve all of its clients. It might need to use several proxies to serve a large number of clients of a video. If these proxies remain non-cooperative, their overall resource utilization, such as memory utilization will be poor. A good scalable streaming system should allow cooperation among the proxies in an ISP in order to improve their resource utilization.

1.3. Scope and Focus

At the beginning of this chapter, we have listed five problems of video streaming over the Internet. The first problem, namely the need for QoS guarantees by the Internet, is outside the scope of this dissertation and has been attacked through standardizing various QoS mechanisms such as IntServ [4], DiffServ [5], and protocols such as MPLS [6] by Internet Engineering Task Force (IETF) [3]. Our research focuses on the specific issues of bandwidth, memory and computational requirements as well as scalability and interactive playback mechanisms.

In the previous section, we have seen some partial solutions to the problems of interactive and scalable streaming systems. In this dissertation, we propose a new proxy based interactive and scalable streaming scheme. We solve several problems not addressed by previous work. Our streaming scheme can provide different interactive playback modes to a large number of concurrent clients while keeping the streaming service fully scalable. As further improvements, we propose an on-demand video segmentation and proxy buffer provisioning scheme and a collaborative proxy peering system. We have developed analytical models to verify the performance of our schemes. We have also run Java simulation programs based on our analytical models to evaluate the performance of our schemes.

1.4. Outline

This dissertation comprises seven chapters, organized as follows:

- Chapter 1 provides the motivation, the problem definition together with a review of previous work, and the focus and scope of this research.
- Chapter 2 presents background materials related to this research.

- Chapter 3 proposes our proxy based constant-bit-rate streaming scheme for VBR-encoded videos and our scalable streaming scheme with constant-bit-rate transmission over the Internet. Performance evaluation of the proposed scalable streaming scheme is also given in this chapter. Our initial streaming scheme, presented in this chapter, does not have interactive playback capability.
- Chapter 4 describes our scalable streaming scheme with interactive playback capability and evaluates its performance.
- Chapter 5 describes and evaluates the performance of an on-demand video segmentation and proxy buffer provisioning scheme. A goal of the latter is to avoid buffer over provisioning at the proxy.
- Chapter 6 presents and evaluates a collaborative proxy peering system. Its goals are to improve proxy buffer utilization and to gain improved performance in on-demand segmentation and buffer provisioning.
- Chapter 7 summarizes our contributions and gives the outline of our future work.

2. Background

2.1. Introduction

The objective of the research described in this thesis is to find a suitable solution for scalable video streaming to a large number of interactive clients over the Internet. We have described previous research work related to our research, in Chapter 1, together with the problem definition. We deal with separate issues in scalable video streaming in Chapters 3, 4, 5, and 6. Each of these chapters also discusses previous research work related to the corresponding issue. In this chapter, we give an overview of the literature related to our research. We describe the Internet and access networks, video encoding, streaming protocols, and streaming system.

2.2. Internet and Access Network

The Internet is the network of networks distributed all over the world. An access network is a local network that allows home users to get connected to the Internet. Figure 2.1 shows two such access networks. A video stream from a streaming server needs to be transmitted through the Internet and through an access network in order to serve a home client. We next briefly describe the Internet in Section 2.2.1 and access networks in Section 2.2.3.

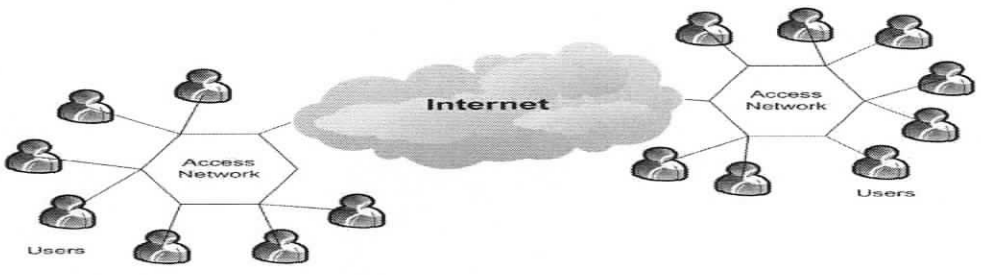

Figure 2.1: Access networks connect users to the Internet

2.2.1 The Internet

Although the Internet's ancestor was a single network, namely ARPANET, in 1969, the present Internet is a network of a large number of autonomous networks. Each autonomous network is called an autonomous system (AS) and is administered by an autonomous administration. Each AS may also consist of several sub-networks or subnets. The Internet is a datagram network; a datagram or a data packet from a source computer enters the Internet at one point and leaves it at another point to a destination computer. [The source and the destination computers are called hosts.] Each data packet travels through the routers in the Internet as an independent entity. Each packet has an Internet Protocol (IP) [38][54] header that contains the source and the destination host addresses, called IP addresses. Seeing the destination address, each router forwards each packet independently to the next router, which should be closer to the destination. Finally, the packet leaves the Internet and reaches the destination host.

Routers in an AS are linked with each other in a mesh-like topology and are controlled as a single administrative domain. Routers inside an AS generally use a link-state routing protocol called Open Shortest Path First (OSPF) [57] to route the packets. An OSPF router floods its link state, which includes its link loading and the cost metrics, such as delay, throughput etc., to other OSPF routers in the same AS. An OSPF router disseminates its link state either at its start-up time or when a change occurs in its link state. At any instant all the routers under the same OSPF scope hopefully have the same link-state information about the whole network. From this link-state information a router knows the network topology and the network constraints. Each router computes a shortest path to every other router in order to route the packets using Dijkstra's shortest path algorithm [40].

Routers and the links in different autonomous systems are controlled by different administrative domains. However, two Autonomous Systems may build a peering relation

to carry each other's packets. Figure 2.2 shows that three ASs may build peering relationships with each other through *border routers*. The Border Gateway Protocol (BGP) [58] is generally used to establish the peering relationship between two ASs. BGP is a distance vector protocol and also may enforce policies agreed by the peering ASs. A BGP router passes a list of ASs it can reach to its neighbour as a path vector. When a BGP router learns about a new AS from a neighbour BGP router, it includes that AS in its own path vector, which it sends to other neighbours. In this way a BGP router knows about a distant AS, which is not its immediate neighbour. Each AS in the path vector is counted as a network hop. A shortest hop count path from an AS to another AS is computed in order to route the packets using the Bellman-Ford algorithm [36].

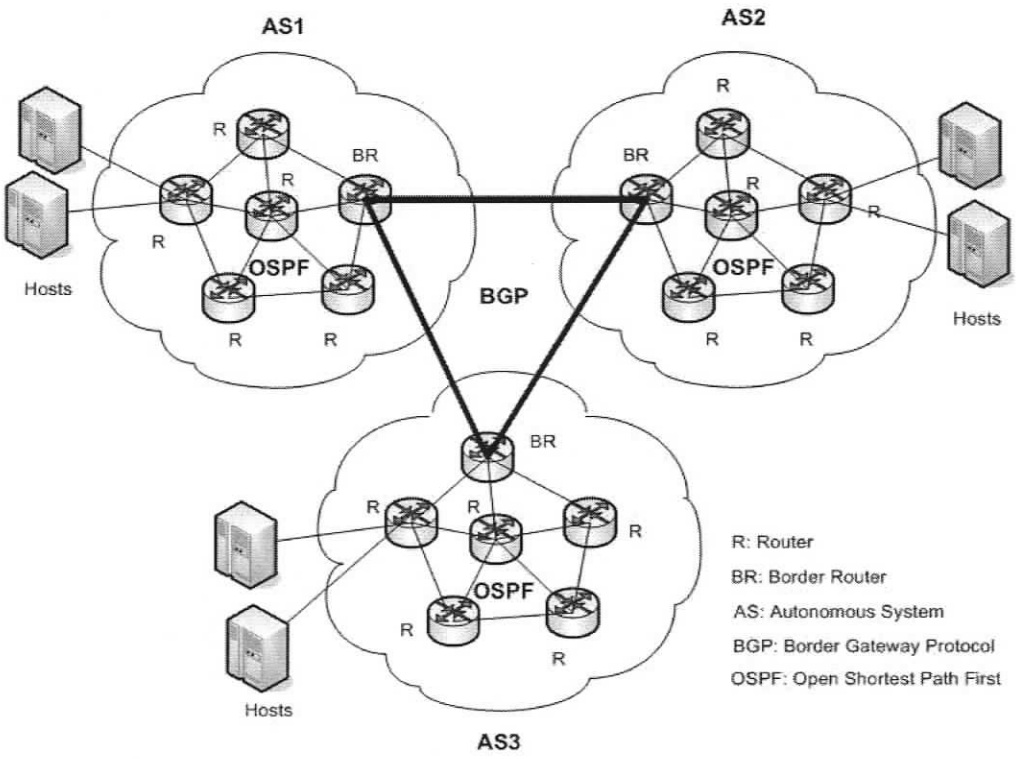

Figure 2.2: Autonomous Systems in the Internet

The routers at the edge of an autonomous network are called the *edge routers* and the routers in the middle of the network are called the *core or backbone routers*. Core routers need to forward relatively more packets than the edge routers. All routers (both edge and core) implement the physical, data link, and network layer protocols of the TCP/IP [56] protocol stack. The Network Layer protocol is called the Internet Protocol (IP) [54] in the Internet. Host computers also implement an additional layer, called *Transport Layer*. Transmission Control Protocol (TCP) [55] is the transport layer protocol for the Internet.

The Internet grew rapidly after TCP/IP [56] became the de-facto protocol standard. Growth continued exponentially, and by the 1990 the Internet had grown to 3000 networks and 200,000 hosts [50]. By 1995, there were multiple backbones, hundreds of regional networks, tens of thousands of local area networks (LANs), and millions of hosts. The Internet's size doubles approximately every year [46]. The Internet has already become a universal network that can connect computers from anywhere in the world. This has created the opportunity to run many applications worldwide. World Wide Web (WWW), e-mail, newsgroups, remote login, and file transfer are among numerous applications that have been successfully running over the Internet for many years.

The number of new applications on the Internet is also growing every year. Internet video-on-demand (IVoD), Internet telephony (VoIP), and Internet videoconferencing are Internet applications with lots of future business potential. These applications need continuous and real time delivery of audio/video data over the Internet, i.e., they need some or all of guaranteed throughput, guaranteed maximum delay, guaranteed maximum variance of delay, and guaranteed minimum packet loss ratio. However, the present Internet was not designed to guarantee any of them.

In the IP layer, no end-to-end connection is established in the network before data transmission starts. Therefore, no bandwidth or throughput guarantee is possible, as

bandwidth cannot be reserved along a path, which does not exist. Individual routers forward individual packet independently. Queues of packets are also built up at the routers independently. A router can even drop packets when its queue is overloaded or the forward link is congested. The normal behaviour of router queues on the Internet is called tail-drop. Tail-drop works by queuing up to a certain maximum number of packets, then dropping all traffic that 'spills over'. This is very unfair, and also leads to a retransmit synchronization problem. The sudden burst of drops from a router that has reached its limit will cause a delayed burst of packet retransmissions, which may overfill the congested router again. Routers may use Random Early Detection (RED) queue management [43] to avoid congestion or queue overflow. RED statistically drops packets from flows before it reaches its hard limit. This causes a congested backbone link to slow more gracefully, and prevents retransmit synchronization by allowing some packets to get dropped sooner, keeping queue sizes low and under control. However, RED may be difficult to tune and stabilize.

From the above discussion, it is clear that the final delivery of a packet as well as its delivery time is not guaranteed. For this reason, the Internet service is called *best-effort datagram service*, which does not guarantee maximum delay or maximum jitter (the delay variance) in packet delivery. Its best-effort nature makes it hard to stream audio/video over the Internet with an acceptable quality.

2.2.2 Quality of Service (QoS) Guarantees in the Internet

Bandwidth, maximum delay, maximum delay jitter and packet loss ratio guarantees are called *Quality of Service* (QoS) guarantees. Though two hosts on the Internet can establish an end-to-end TCP connection to transfer data as a byte stream, this does not yield any QoS guarantee by the network. Inside the network a TCP connection has no significance. However, for audio/video streaming, end-to-end QoS guarantees are needed from the network. The Internet Engineering Task Force (IETF) has proposed Integrated Services (IntServ) [4] for this purpose. IntServ uses the Resource Reservation Protocol

(RSVP) [48][49] to reserve resources [primarily link bandwidth] on a path in order to ensure end-to-end QoS guarantees for a connection. IntServ and RSVP are not scalable though they together can ensure end-to-end QoS guarantees in the network. Routers need to maintain soft state for each connection that passes through them. Since the number of connections passing through a core router is huge, this IntServ/RSVP solution has not found much acceptance in the industry.

IETF has proposed another mechanism called Differentiated Services (DiffServ) [5]. DiffServ has re-defined the type of service (TOS) bits in the IP header as the differential service code point (DSCP) and classifies IP datagrams into several classes. DiffServ's defined classes are expedited forwarding (EF), assured forwarding (AS)-1 to AS-4, and best effort (BE). The packets from different classes get differential treatments at the routers. DiffServ works within the boundary of an AS. Using the service level agreement with a customer, an edge router of an AS polices, shapes, and marks the customer traffic before it enters into the network. Edge routers use DSCP bits to mark a packet. Class level treatment or QoS guarantees are given to each marked packet when it travels through the core routers. Routers do not need to maintain soft-states for the connections. In fact, there is no connection in DiffServ. DiffServ is a scalable mechanism and it guarantees class level, relative QoS. However, end-to-end QoS guarantees can be achieved by using DiffServ with a protocol that establishes an end-to-end path.

IETF has proposed another protocol namely Multi-protocol Label Switching (MPLS) [6][7] to ensure QoS guarantees through the network. In MPLS networks, routers are called Label Switched Router (LSR) and each packet carries a label called its MPLS label. QoS metrics about the MPLS enabled network are maintained at the LSRs. A QoS guaranteed path from a source to a destination is computed by the OSPF routing protocol using the QoS metrics. An MPLS path or connection is then established over the network through MPLS label bindings at each LSR on the path. At each LSR, each path is bound to two labels; one is incoming and the other one is outgoing. The label bindings of all the

MPLS paths passing through an LSR are saved in a table in the LSR. All the incoming packets on an MPLS path come with the incoming label. Verifying the incoming label of a packet, an LSR decides its outgoing link and finds the outgoing label from the table. The LSR substitutes the incoming label with the outgoing label and forwards the packet to the next LSR. The above label switching and packet forwarding mechanism continues until the packet reaches the destination. Since an LSR forwards packets only after examining their MPLS labels, it does not need to examine the packet headers, i.e., packet forwarding can be very fast. MPLS specifies ways that Layer 3 traffic can be mapped to connection-oriented Layer 2 transport mechanisms like ATM [41] and frame relay [72]; it adds a label containing specific routing information to each IP packet and allows routers to assign explicit paths to various classes of traffic. A variant of MPLS called Generalized MPLS (GMPLS) [59] has also been proposed to use switched optical-optical-optical (OOO) routers at the core of the next generation Internet in order to reduce delay and jitter to a great extent.

2.2.3 Access Networks

Corporations and the universities manage their own private IP networks, called intranets, and connect them to the public Internet through gateways. These gateways are often firewall protected in order to protect internal communications from outside attacks. Users in a corporation or in a university can communicate with each other using their intranet safely. They can also access the public Internet from their intranet. Home users, on the other hand, do not have such luxury. They have to subscribe to an access network. An access network is a network that connects individual home users to the public Internet. An access network can be based on Public Switched Telephone Network (PSTN), Cable TV, Digital Subscriber Loop (DSL) or Passive Optical Network (PON). A PSTN dial-up connection uses the normal telephone local loop and a dial-up modem to connect a household computer to the Internet. The maximum possible speed of a dial-up connection is very poor, perhaps 56 kbps. The Data Over Cable Service Interface Specification (DOCSIS) [30][31][32] has been developed and cable networks are now used for two-

way Internet communications using DOCSIS. The downstream bandwidth range in DOCSIS is 30.3 Mbps to 42.9 Mbps. DSL's bandwidths vary from 128 kbps to 51 Mbps. PON's bandwidths vary from 155.52 Mbps to 622.08 Mbps. It is clear that DOCSIS (over Cable TV network), DSL, and PON are suitable access network technologies to deliver video streams to the home users. We briefly describe them in the Appendix in Sections A.1, A.2, and A.3 respectively.

2.3. Video Encoding

In general, video files are huge. For example, five minutes of uncompressed video will require about 1 gigabyte of storage. For this reason, when a video is prepared for storage or for streaming, the video file is generally encoded to compress it. In video streaming, compressed files are transmitted from the server in a steady stream and are decompressed in real-time by the media player for media playback. Several video encoding standards are available today, such as H.261, H.263, and H.264 by the International Telecommunication Union-Telecommunication (ITU-T) [73], and MPEG-1, MPEG-2, and MPEG-4 by the Motion Picture Experts Group (MPEG) [74].

H.261 [10] and other H.26x standards were designed for videophone, videoconferencing, and other audiovisual applications over ISDN telephone lines. MPEG-1 [11] was originally designed for VHS quality video on CD-ROM for quarter screen TV (352x240) at 30 frames/sec or at 1.5 Mbps. MPEG-2 [12][13] was designed for digital TV broadcasting with a target bit rate of 4 Mbps to 9Mbps. It can also be used for HDTV with a bit rate 80 Mbps. MPEG-4 [14][15] was designed for web and mobile delivery with low bandwidth connections from 64 kbps to 4 Mbps. MPEG standards have become the de-facto standards for the industry; we describe the MPEG-2 standard briefly in the Appendix in Section A.4.

2.3.1.1 Scalable Encoding

A scalable-encoded-video stream consists of a base sub-stream and several enhancement sub-streams. The base sub-stream alone provides low-resolution video playback. An enhancement sub-stream alone cannot produce a video playback. However, a video playback can be improved by combining one or more enhancement sub-streams with the base sub-stream. Different combinations of base sub-stream and enhancement sub-streams are used in different situations. Scalable encoding is necessary for error resilience and simulcasting the stream with different bandwidth constraints, to different devices, with different qualities/resolutions, and with different interactive playback modes. MPEG-2 supports 4 types of scalable encoding: data partitioning, signal-to-noise-ratio (SNR), spatial, and temporal.

Data Partitioning uses a single-layer encoder. Discrete Cosine Transformed (DCT) data from the encoder is partitioned into a base sub-stream and an enhancement sub-stream. The base sub-stream carries only the first few low frequency DCT coefficients, including the DC coefficient, and the enhancement sub-stream carries the remaining DCT coefficients, i.e., high frequency coefficients.

In SNR scalable encoding, the base sub-stream carries the frames with low bit-rate and low quality picture. Base sub-stream frames are produced using higher quantization values to reduce the bit-rate. The enhancement sub-stream carries additional data for each frame in the base sub-stream to enhance the quality of the picture.

In spatial scalable encoding, the base sub-stream carries the low frequency component (approximation part) of each frame and an enhancement sub-stream carries a higher frequency component (detail part) of the corresponding frame. This is also known as sub-band encoding. Wavelet encoding is a particular type of sub-band encoding.

Frames from selective temporal locations in the original stream are kept in the base sub-stream in temporal scalable encoding. The remaining frames of the original stream are kept in the enhancement sub-stream. The base sub-stream yields low frame rates and low temporal resolution. By adding the enhancement sub-stream to the base sub-stream, the frame rate is increased to provide better temporal resolution.

2.3.2 Constant-bit-rate (CBR) and variable-bit-rate (VBR) Encoding

Video encoding can be either constant-bit-rate (CBR) or variable-bit-rate (VBR) [45]. CBR encoding enforces a constant bit rate over a group of pictures (GOP) regardless of the complexity of the video sequence. CBR encoders enforce different quantization scales for each picture type. To generate a constant bit rate stream, a rate buffer is placed at the output of the encoder. The rate buffer is drained at a constant rate; in order to ascertain that the buffer does not overflow or underflow, the buffer occupancy level is used as feedback to control the quantization scale. Thus quantization is much coarser when the sequence is action-filled, and the quality of the resulting pictures suffers as a result. For a CBR encoded stream the channel bit rate is equal to the encoding rate. Since the bit rate is constant, a static bandwidth allocation scheme can be used.

Video is inherently variable bit rate. A video of a static image like a bowl of fruit needs a much lower bit rate than a video of a hockey game. However, CBR encoding reduces the degree of freedom of the encoding task. CBR encoding also reduces the quality of those video intervals like boxing matches, car races, etc which are action-filled.

In order to get constant quality video, VBR encoding is preferred to CBR encoding. The quantization scale in VBR encoding is simply kept at a constant value and no rate buffer is placed at the output of the encoder. The bit rate over GOP therefore varies with the complexity of the video interval. Since the encoding rate is variable, a dynamic bandwidth allocation scheme is optimal to transmit a VBR encoded stream. VBR

encoded streams also have frequent traffic bursts. For this reason, one needs a high peak bandwidth for transmitting a VBR encoded video.

2.4. Streaming Protocols

A streaming client can request an audio or video stream from a streaming server using the Real-Time Streaming Protocol (RTSP) [2]. RTSP allows a client to control media playback, though RTSP itself does not transfer media data from the server to the client. Actual data transmission takes place out-of-band in a different protocol called Real time Transport Protocol (RTP) [1]. We briefly describe RTSP in Section 2.4.1 and RTP in Section 2.4.2.

2.4.1 RTSP

The Real time Streaming Protocol (RTSP) establishes and controls a single or several time-synchronized streams of continuous media such as audio and video. RTSP has been described in RFC-2326 [2]. The set of streams to be controlled by RTSP is defined by a presentation description. We describe the presentation description in the Appendix in Section A.5.1. In the presentation description, a RTSP URL identifies each media stream that is individually controllable by RTSP. We describe the RTSP URL in the Appendix in Section A.5.2.

A RTSP server maintains the presentation description in a file. A RTSP client can request a presentation description via HTTP or some other means. The client issues a SETUP request to the server in order to establish a RTSP session. The client includes a RTSP URL from the presentation description in the SETUP request. The RTSP server upon the completion of a successful SETUP request, i.e., after sending an OK response, establishes a session and labels it by a session identifier. The Client gets the session identifier in the SETUP response message. A RTSP session is a server entity; it is created, maintained and

destroyed by the server. The session exists until timed out by the server or explicitly removed by a TEARDOWN request. A RTSP session is also a stateful entity; a RTSP server maintains an explicit session state machine where most state transitions are triggered by client requests. We describe RTSP session states in the Appendix in Section A.5.5.

A session can have zero or more associated media streams. At any time, the client can add or drop a media stream from a session. The client issues another SETUP request to the server enumerating the session identifier and the RTSP URL of a media stream in order to add the media stream to a session. The client issues a TEARDOWN request enumerating the session identifier and the RTSP URL of a media stream in order to drop the media stream from a session.

An RTSP server uses the session to aggregate control over multiple media streams. A client can issue a PLAY request enumerating the RTSP URL of a media stream to start its playback. A client can issue a TEARDOWN request enumerating the RTSP URL of a media stream to stop its playback. Each successive client request of a session carries the session identifier and triggers the state transition of the session. RTSP has several request methods other than SETUP, PLAY, and TEARDOWN. We describe all RTSP request and response message formats and request methods in the Appendix in Sections A.5.3 and A.5.4 respectively.

2.4.2 Real-time Transport Protocol (RTP)

The Real-time Transport Protocol (RTP) has been described in RFC-1889 [1]. RTP provides end-to-end transport services appropriate for applications that transmit real-time data, such as audio and video, over the Internet. RTP can transport real-time data over unicast as well as multicast networks. RTP does not do resource reservation and does not guarantee any quality-of-service (QoS). It is designed to work in conjunction

with an auxiliary control protocol, RTP Control Protocol (RTCP), to get feedback on quality of data transmission and information about participants in the ongoing RTP session. Though UDP and IP are the recommended transport and network layer protocols for both RTP and RTCP, they are application layer protocols and are independent of the underlying transport and network layers. RTP transfers a chunk of real-time data, encapsulating it into an RTP packet. We describe the RTP packet, MPEG stream encapsulation into RTP packets, and RTCP packets in the Appendix in Sections A.6.1, A.6.2, and A.6.3 respectively.

2.5. Streaming System

Streaming means delivery and playback of audio and video data in real time. It is different from downloading. In downloading, compressed media files are kept on the web server. When a browser requests a media file through a HTTP GET request, the web server sends the media file to the browser in its HTTP response message. The Browser waits to complete downloading the whole media file before it launches the media player. Media packets arrive asynchronously, are reassembled and written onto disk. Once the downloading is finished the browser launches the media player and passes the media file to it for playing. Video files are often large and so the downloading time will be long too.

Once the media player is launched it decompresses the media file and plays it on the client's computer. Only stored media files can be downloaded; live streams cannot be downloaded; and the copyright protection of a downloaded media file is an issue that deeply concerns content owners.

The Streaming concept has been developed to transfer media data over the network, avoiding the problems that the downloading mechanism faces. In streaming, audio and video data from a streaming server are transferred over the network in such a way that they can be processed as a steady and continuous stream at the client. A client does not

need to wait for a long time to finish downloading the whole media file to start the playback. Instead, it can start playback when it starts receiving the stream. Streaming data are time synchronized and played immediately according to synchronization time at the client. A client does not write streaming data on its disk and can throw away the parts of the streaming data that has already been played. A streaming server can transmit both live and stored media; copyright is not violated if streaming does not write the media file on the client's disk. Though streaming is considered better for transmitting media data over networks than downloading, it has its own limitations. In streaming, smooth media playback highly depends on the encoding bit rate and the available bandwidth between the streaming server and the client. If the available bandwidth is less than the encoding bit rate, smooth playback is not possible. For example, CD quality audio requires at least 1.5 Mbps, and MPEG-2 encoded media requires at least 4 Mbps bandwidth.

A simple streaming system consists of a streaming server or server farm and a large number of streaming clients. A streaming client requests a stream from a streaming server using the RTSP protocol. A streaming server transmits media data to the clients using the RTP and RTCP protocols. In Streaming, media delivery is controlled from the server using the RTSP protocol. The streaming server delivers the media data to the streaming player in real time. Typical live streaming content could be lectures, sports or entertainment events, news events, videoconference, and ceremonies. Good examples of stored media streaming are video-on-demand, full motion video etc. RealServer from RealNetwork [68], QuickTime Server from Apple [67], and MS Media Services from Microsoft [69] are examples of good streaming servers.

After receiving media content a media player plays it in real time. RealPlayer from RealNetwork, Quicktime Player from Apple, and Windows Media Player from Microsoft are few examples of some popular media players available.

2.5.1 Scalable Streaming System

The number of streaming clients simultaneously served by a streaming system is called its throughput. The amount of time elapsed from when a client request arrives at the system until the actual playback is initiated by the system for the request is called the *start-up latency*. The total cost for resources at the streaming server and on the transport network can be divided by the throughput in order to find cost per stream. A scalable streaming system is one where the cost per stream and the start-up latency do not increase with a significant increase in its throughput.

There are two ways to make a streaming system scalable: one way is by using a streaming proxy between the server and the clients and the other way is by using a multicast enabled network between the server and the clients.

2.5.1.1 Multicast-Based Scalable Streaming Scheme

If a multicast enabled network is available between the server and the clients a single multicast stream can be used to serve a large number of synchronous clients together, which will increase the throughput without increasing the cost per stream and the start-up latency. Different client requests for the same stream generally come at different times, i.e., they are asynchronous, which makes it difficult to serve them together with a multicast stream. There are several solutions for this problem. They are Batching [63], Patching [62], and Merging [64].

Batching batches all the requests for the same streaming media in a batching interval and serves them together by a single multicast transmission at the end of the interval. Batching adds latency equal to the batching interval.

Patching does not delay the transmission until the end of a batching interval. It starts a multicast stream immediately after receiving a request. If a later request comes before a patching threshold (interval) it allows the later client to cache the streaming data from the on-going multicast transmission and receive the missing part of the streaming data or the patch on a separate unicast channel. Patching does not add latency; however, a later-arriving client needs two channels, one multicast and one unicast, and a large buffer to prefetch the multicast stream.

Merging merges two separate multicast streams into a single multicast stream if the time difference between the streams does not cross a preconfigured merging threshold. It merges two stream by slowing down the earlier stream and speeding up the later stream so that they become synchronized within the threshold time. Though merging seems better than batching and patching, it is difficult to implement.

2.5.1.2 Proxy Based Scalable Streaming Scheme

Streaming proxies [65] can relay RTSP and RTP traffic between the server and its clients. Streaming proxies reside near the clients and intercept client requests for a video. Due to the large size and copyright protection of a media file, a streaming proxy does not cache a media file. Rather it forwards the client request to the server. The Proxy receives the server stream and transmits it to the client. The Proxy generally buffers the server stream in a window of few seconds. If another client needs the same stream at almost the same time, the proxy can relay the stream from the buffer, which increases throughput without increasing the cost per stream and the start-up latency.

2.6. Chapter Summary

In this chapter, we have described the Internet, access networks, video encoding, streaming protocols, and streaming system in order to provide background knowledge related to our research work.

3. A Scalable Multimedia Streaming Scheme

3.1. Introduction

Streaming audio/video over the Internet requires timely delivery of streaming data so that it can be processed as a steady and continuous stream at the receiver end. However, timely delivery of streaming data is difficult over the Internet due to network latency and jitter. Large network latency causes large start-up delays and large network jitter inserts unwanted pauses in the playback. Each stream needs a large network bandwidth, e.g., an MPEG-2 [12][13] stream needs mean bandwidth of 4 Mbps. For a streaming server even an OC-48 (about 2.5Gbps) connection is often not enough to serve thousands of clients simultaneously, e.g., the audience for the latest popular movie. A video streaming session generally lasts for a long time, in the order of an hour. It also needs a large portion of the generally limited I/O bandwidth at the server during this long session. For these reasons, a streaming server alone cannot provide a satisfactory scalable streaming service. Therefore, proxy servers need to be deployed to address the scalability problem.

However, it is often infeasible to cache an entire streaming media file at a proxy, for two reasons. First, streaming media files are typically large, on the order of a gigabyte. Secondly, for copyright and for business and security reasons streaming-content owners do not want the proxy or the client to cache the entire media file. These issues make a streaming service hard to scale using conventional proxy servers.

Media encoding techniques, such as MPEG-2 and MPEG-4 [14][15], are used to compress large media files. Compressed audio/video files are sent from the server in a steady stream and decompressed in real-time by the media player during the playback. A media server can easily stream a CBR-encoded video at a constant bit rate equal to its

encoding bit rate. However, CBR encoding reduces the degrees of freedom for the encoding task and the quality of video intervals such as in a boxing match, which is full of action. For these reasons, VBR encoding, which provides constant quality video intervals, is preferred to CBR encoding. Frequent traffic bursts in a VBR-stream create new problems. A media server cannot stream a VBR-encoded video at a constant bit rate; rather it needs to stream the video at variable bit rates. If the peak bandwidth is not allocated to a VBR-stream, frequent traffic bursts insert frequent pauses in the media playback. Peak bandwidth allocation, however, wastes network bandwidth and makes a streaming server capable of streaming only a few VBR streams concurrently. Therefore, VBR streams make it more difficult to achieve scalability in a streaming scheme.

In this chapter, we propose a proxy based CBR-transmission scheme to smooth out network jitter and traffic bursts of VBR-encoded videos and to enable near- mean bit rate transmissions. We also propose a scalable streaming scheme to stream stored videos to a very large number of clients over the Internet, using CBR transmission. We assume that the streaming proxies are deployed close to the clients and that the Internet connects those proxies to a streaming server via channels with acceptable QoS guarantees. We also assume that the connections between the proxies and the clients can be Digital Subscriber Line (DSL), broadband Cable Network, or Passive Optical Network (PON). We assume unicast transmission capability both between the server and the proxies and between the proxies and the clients. Our aim is to make the transmission scheme highly scalable through sharing a server-to-proxy stream among as many clients as possible and by reducing the peak bandwidth requirements on both the server-proxy and the proxy-client paths. We achieve our goals while keeping the server complexity and the client requirements to a minimum.

We use a *prefix buffer* at the proxy to store some initial frames or the prefix of a video in order to hide the network delay between the server and the proxy and to assist in *work-ahead smoothing*. We deploy a new work-ahead smoothing technique in order to enable

CBR-transmission from a server as well as from a proxy, overcoming both jitter and VBR traffic burst problems. We try to maintain the CBR-transmission close to the mean bit rate of the video in order to reduce the peak bandwidth requirement significantly. We use a *smoothing buffer* that with the help of above prefix buffer implements the work-ahead smoothing. We use the smoothing buffer not only to smooth out VBR traffic but also to enable many clients to share the same server stream, to help make the streaming service scalable. We reduce the buffer space requirement at the proxy by sharing a smoothing buffer among many clients. We use the smoothing buffer instead of the prefix buffer to enable many clients to share the same server stream, in order to make our scalable streaming scheme effective for both CBR- and VBR-encoded videos. We perform work-ahead smoothing operations to eliminate just-in-time arrival requirements from the network. Our scheme needs neither multiple channels, nor a very large buffer, nor complicated processing at the clients. Therefore, a thin client, such as a Personal Digital Assistant (PDA), should be able to play a video stream using our streaming scheme. All of the above features highlight the significant improvements of our work over those of some previous work in this area. The main contributions of our scheme are as follows.

- It smoothes out network jitter and traffic bursts of VBR-encoded videos, effectively hiding network latency. As a result clients always get quick and hiccup free playback.
- By sharing the same server stream and proxy buffers among many clients, our scheme remains highly scalable.
- It allows a streaming server and a streaming proxy to transmit a VBR-encoded video at a constant bit rate close to the mean encoding bit rate; reduces the bandwidth requirement on both the server-proxy and the proxy-client paths for each stream and enables a streaming system to transmit more streams concurrently, i.e., improves scalability.
- The use of a smoothing buffer to share the same server stream among many clients makes our scheme effective for both CBR- and VBR-encoded videos.

- Our scheme minimizes client requirements, i.e., a thin client with only a single channel and a small buffer should be able to share a server stream with many other clients.

The rest of this chapter is organized as follows. We describe related research work in Section 3.2, and our proposed CBR-transmission and scalable streaming schemes in Section 3.3. In Section 3.4, we present our simulation results. In Section 0, we conclude the chapter.

3.2. Related Work

Boroczky et al. [19] propose a technique to statistically multiplex several VBR-encoded concurrent streams into a larger, less bursty stream, hence closer to a CBR stream, in order to minimize the server bandwidth requirement. This technique effectively reduces the overall streaming bandwidth requirement only when the concurrent streams have statistically varying frame sizes. Therefore this technique is not applicable when a server streams a single video to a large number of clients concurrently. There are several online smoothing techniques [20][21][22][75] that pre-fetch the prefix of a video into a buffer at the proxy or at the client, and convert the VBR traffic into several consecutive runs of different CBR traffics, thus reducing the peak rate as well as the variability of data rates of a VBR-encoded stored media. However, these smoothing techniques do not reduce the peak bit rate down to the mean bit rate and do not eliminate rate variability completely; and they add server complexities.

Sen et al. [17] use a proxy with prefix and work-ahead smoothing buffers to help mask network latency and jitter and to do online smoothing of VBR traffic bursts. They effectively do online smoothing only on the proxy-client path but not on the server-proxy path. Neither the prefix buffer nor the smoothing buffer is used to enable stream or buffer sharing among the clients. Each client needs a separate smoothing buffer at the proxy;

hence, the scheme requires a huge amount of buffer space at the proxy to stream a video to a large number of clients. It also needs a separate server stream for each client due to its purely non-scalable characteristics. Since no smoothing measures are taken on the server-proxy path, the bandwidth requirement on this path remains high and variable, which complicates the bandwidth allocation process at the server.

Wang et al. [18] use a proxy with *prefix buffers* to mask network latency and to enable several clients to share the same server stream in order to make a streaming service scalable. Their streaming scheme can stream only CBR-encoded videos. However, due to the absence of a smoothing buffer, VBR-encoded videos cannot be streamed in their streaming scheme. Their streaming scheme caches the *prefix* or the first few frames of a video in a prefix buffer at the proxy. After intercepting a client request, the proxy immediately starts streaming from the prefix buffer and requests the *suffix* or the remaining part of the video from the server. They have assumed that the proxy receives the suffix data just in time when it finishes prefix streaming. If more requests come for the same video before it finishes the prefix streaming, the proxy enables a single suffix sharing among these clients. They consider just-in-time arrival of streaming data at the proxy, an assumption whose validity cannot be guaranteed for the Internet. Also, they do not use any work-ahead smoothing operation to handle network jitter. In their scheme, a client often needs to receive both prefix and suffix data simultaneously, i.e., needs multiple channels. This is a requirement that may not be economically feasible in many cases, such as current Digital Subscriber Loops (DSLs). In their scheme, a client also needs a large buffer to buffer the suffix while the prefix is being played.

3.3. Proposed Scalable Streaming Scheme

In this section we describe our scalable streaming scheme. We start with the system model and then describe our CBR transmission and scalable streaming schemes. We also

compute the prefix buffer size, smoothing buffer size, and the average streaming cost required to transmit a video using our scalable streaming scheme.

3.3.1 System Model

We propose a streaming model, where a server uses one or more proxies to stream multiple videos to a very large number of clients and to provide quick and hiccup free media playback. We reduce the peak bandwidth requirements of both the server-proxy and the proxy-client paths for VBR-encoded stored videos. We also make the transmission scheme scalable through sharing a server-to-proxy stream among as many clients as possible. We want to achieve our goals while keeping the server complexity and the client requirements as low as possible. We also assume the server and the proxies are connected via the Internet with varying degrees of QoS guarantees and with only unicast enabled. The best effort IP datagram service is not suitable for our model. Our model needs guaranteed bandwidth, and a limit on the maximum allowable delay and on the jitter between the server and the proxy, i.e., we need absolute QoS guarantees in the Core Network. [We can expect such QoS service guarantees from MPLS enabled networks.] Though a high level QoS guaranteed communication between the server and the proxy has a higher cost than that of a best effort communication between the server and the proxy, we are saving lots of ISP costs by minimizing the number of independent streams between the server and the clients. We do this by pre-fetching the prefix of a video and then sharing the prefix and a single suffix stream of that video among as many clients as possible. For example, an ISP may use a dedicated 10Mbps channel between Victoria and Los Angeles for streaming video that can serve hundreds of clients. Since different clients want to watch the same movie at different times, multicasting a single server stream alone will not work. Moreover, multicast enabled networks are not widely available. For these reasons, we propose a proxy based scalable streaming scheme on unicast networks.

We also assume that the path between the proxy and the client uses a QoS guaranteed link, such as DSL, Cable Network, Gigabit-Ethernet (GigE), or PON. These assumptions are realistic for currently deployed and soon-to-be-deployed technology in North America and the Asia Pacific regions. Figure 3.1 shows a simplified diagram of our model.

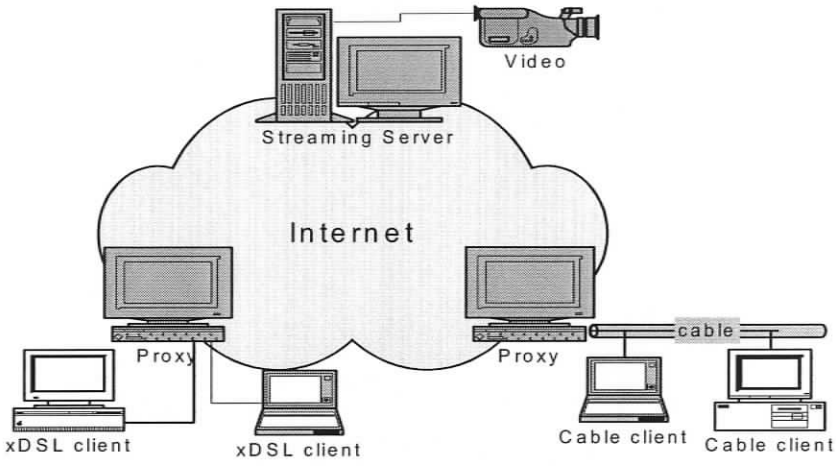

Figure 3.1: System model for streaming

We propose to divide a video into two parts: prefix and suffix. The prefix of a video is the first few seconds of the video, and the suffix is the remaining part. In our proposal, a proxy caches the prefix in a *prefix buffer* in order to overcome network latency and to help in work-ahead smoothing. The proxy intercepts a client's request to the server, and starts streaming from the prefix buffer immediately, i.e., it overcomes the often-large network latency between the server and the proxy. The proxy allocates a smoothing buffer, requests the suffix data from the server, and stores the incoming suffix data in the smoothing buffer. The smoothing buffer with the help of the prefix buffer does *work-ahead smoothing*. In work-ahead smoothing we receive and buffer the streaming data earlier than it is needed for playback. We do this in order to overcome both jitter and

VBR traffic burst problems and to enable near-mean-bit-rate transmission from both server and proxy.

The proxy starts streaming from the smoothing buffer when the streaming from the prefix buffer ends. We use a playback buffer at the client in order to pre-fetch the prefix data before starting playback. [The playback buffer also works as the smoothing buffer at the client after it starts media playback.] We assume the smoothing and the playback buffers are circular buffers, i.e., the read and the write pointers are wrap-around. The client waits to start media playback until it finishes pre-fetching prefix data from the proxy. The client temporarily buffers the work-ahead suffix data coming from the proxy in the playback buffer, in order to get smooth playback. Figure 3.2 shows the various buffers used at the proxy and at the client.

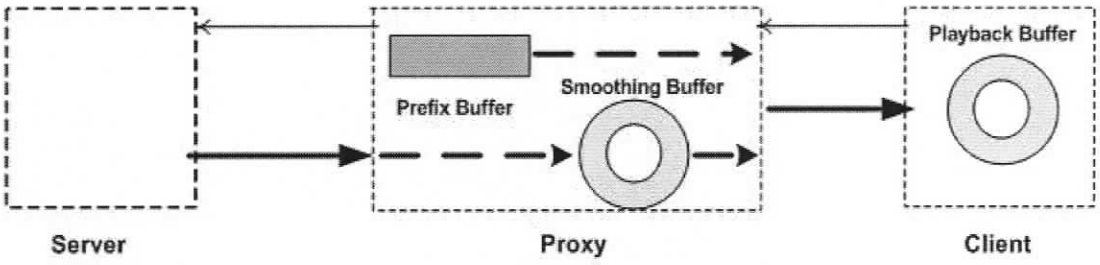

Figure 3.2: Buffers at the proxy and the client

3.3.2 CBR-transmission Scheme

We introduce a new work-ahead smoothing technique in order to overcome both jitter and VBR traffic burst problems and to enable constant and near mean bit rate transmission of VBR streams from the server as well as from the proxy. We call it the *CBR transmission scheme*. CBR transmission from a server or proxy may be at a rate higher or lower than the mean bit rate of a video. This transmission rate depends on the available bandwidth. If the transmission rate is higher than the mean bit rate, there will be less data to store in the prefix buffer but more data to buffer in the smoothing buffer, and

conversely if the transmission rate is lower. Therefore, the ratio between the transmission rate and the mean bit rate of a video, which we call the *rate-ratio* r, controls the amount of data that must be stored in the prefix buffer and in the smoothing buffer. Higher network latency, network jitter and traffic bursts increase the amount of data required in both the prefix buffer and the smoothing buffer. We compute the traffic bursts of a video from its frame statistics. We compute the prefix and the smoothing buffer sizes in terms of maximum network latency, network jitter, traffic burst, and rate-ratio, in Section 3.3.4 and Section 3.3.6, respectively.

We prefer CBR-transmission close to the mean encoding bit rate in order to reduce the peak bandwidth requirement. However, near-mean-bit-rate reception as well as network jitter causes a shortfall of media data during a bursty interval of a VBR-encoded video. These shortfalls cause hiccups in the playback. Our smoothing technique finds the maximum amount of such shortfalls of video data at the decoder when the server transmits the video at a fixed rate. The proxy needs to pre-fetch this maximum amount of video data before starting the playback, to ensure that the shortfall will not occur during the playback, i.e. to eliminate the hiccup from the playback, hence, the dependency of prefix data size on transmission rate.

Using Lemma 1 below we can find the amount of data required to pre-fetch in order to do work-ahead smoothing of the corresponding traffic bursts of a video. Lemma 2 suggests that we can also reduce this amount by increasing the video transmission rate, if there is enough network bandwidth to support the increased transmission rate.

Lemma 1: If R is the set of all positive real numbers, U is the set of all positive integer numbers, $f_j \in U$ is the size of the j^{th} frame of a video with total of n frames, and $f_{mean} \in R$ is the average size of those n frames, there exists a least number of

bits $D_{least} \in U$ to pre-fetch for a given transmission rate-ratio $r \in R$, for which the inequality $\forall j\left[(j \in (1,...,n)) \Lambda \left(\left(\sum_{k=1}^{j} f_k\right) \leq (j \times r \times f_{mean} + D_{least})\right)\right]$ holds.

Proof: The mean frame size is $f_{mean} = \frac{1}{n}\left(\sum_{j=1}^{n} f_j\right)$. The relation between the mean bit rate b and f_{mean} is $b = f_{rate} \times f_{mean}$, where f_{rate} is the frame rate of the video in frames/second. Therefore, in every frame interval $r \times f_{mean}$ bits of video data are transmitted at a rate-ratio r. In j frame intervals $j \times r \times f_{mean}$ bits of video data are transmitted at a transmission rate $r \times b$ bits/second. Hence, the bandwidth requirement for the above is $r \times b$ bps. Let $F_j = \sum_{k=1}^{j} f_k$. We can say that the above inequality holds for any $j \in (1,...,n)$ and for a $D \in U$, if $(F_j - j \times r \times f_{mean}) \leq D$ is true. We can find the least integer number $D_{j,least}$ such that $(F_j - j \times r \times f_{mean}) \leq D_{j,least}$ holds. And it is obvious that $\forall j\left[(j \in (1,...,n)) \Lambda \left(\left(\sum_{k=1}^{j} f_k\right) \leq (j \times r \times f_{mean} + D_{least})\right)\right]$ is true if $D_{least} = \max(D_{j,least})$ for $j = 1,2,...,n$. Figure 3.3 pictorially presents the validity of Lemma 1.

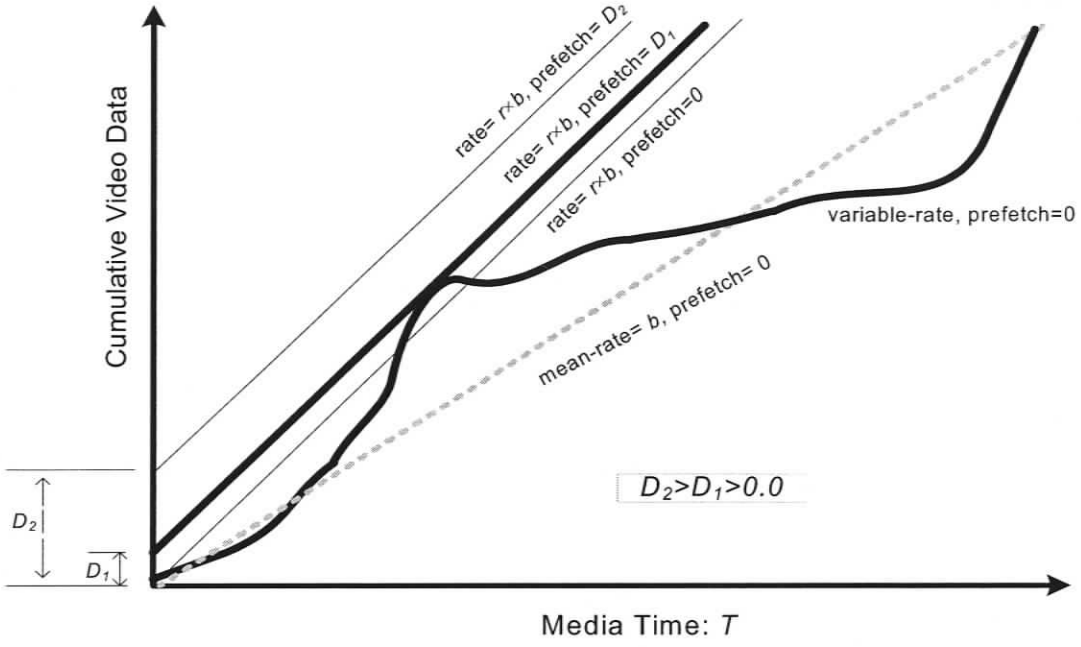

Figure 3.3: Shortfall avoiding using a fixed transmission rate higher than the mean bit rate with a different amount of pre-fetched data.

Lemma 2: There exists a least rate-ratio $r_{least} \in R$ for a given number of pre-fetch bits $D \in U$ for which the inequality $\forall j\left[\left(j \in (1,...,n)\right) \Lambda \left(\left(\sum_{k=1}^{j} f_k\right) \leq \left(j \times r_{least} \times f_{mean} + D\right)\right)\right]$ holds, i.e., the video data can be transmitted smoothly using $r_{least} \times b$ bps of bandwidth.

Proof: We can say that the inequality holds for any $j \in (1,...,n)$ and for a $D \in U$, if $\left(F_j - D\right) \leq j \times r \times f_{mean}$ is true. We can easily find the least rate-ratio $r_{j,least}$ such that $\left(F_j - D\right) \leq j \times r_{j,least} \times f_{mean}$ is true. Now, it is obvious that $\forall j\left[\left(j \in (1,...,n)\right) \Lambda \left(\left(\sum_{k=1}^{j} f_k\right) \leq \left(j \times r_{least} \times f_{mean} + D\right)\right)\right]$ holds if $r_{least} = \max(r_{j,least})$,

where $j = 1,2,...,n$. Therefore, it is proved that for any given number of pre-fetch bits D there exists a least rate-ratio r_{least} such that the above inequality holds.

Figure 3.4 pictorially presents the validity of Lemma 2.

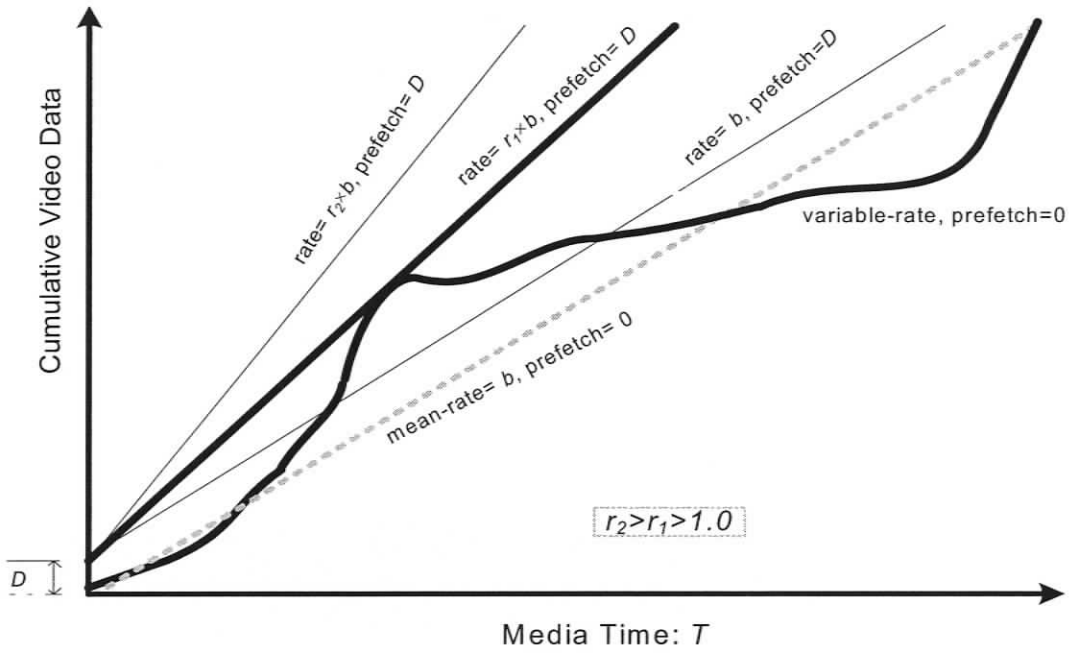

Figure 3.4: Shortfall avoiding using transmission rate higher than the mean bit rate with a fixed amount of pre-fetched data

Parts of the video data may arrive ahead of their decoding times due to the above pre-fetching and/or increased transmission rate. We use our smoothing buffer to hold those work-ahead data temporarily.

3.3.3 UniSMerge: A Scalable Streaming Scheme

We introduce a new scalable streaming scheme that uses our CBR-transmission scheme. We use the same smoothing buffer, which is used to implement CBR-transmission at the

proxy, to enable many clients to share a server-proxy stream. When a new request arrives for a running video, the proxy immediately starts streaming from the prefix buffer. The proxy does not send a new suffix request to the origin server, if the new request can get the suffix data from a previously allocated smoothing buffer. This is possible if the first byte of the suffix data is still available at that smoothing buffer at the time when the proxy finishes the prefix streaming to the new request. In that case, two unicast suffix transmissions from the server are merged into one unicast suffix transmission. For this reason, we call our scheme *Unicast Suffix Merging (UniSMerge)*. We have introduced this new idea of using a circular buffer for merging several server-side suffix transmissions (both CBR and VBR) into a single transmission.

It is obvious that the larger the size of the smoothing buffer the larger the sharing interval. Therefore, by allocating a large smoothing buffer we can make a streaming service more scalable. Also, part of the smoothing buffer is used for work-ahead smoothing, described in Section 3.3.1 and Section 3.3.2. For this reason, the effective sharing interval of a video is always less than its allocated smoothing buffer interval/window. The effective sharing interval of a video can be found when the size of the smoothing buffer, the maximum traffic burst, network jitter, network delay, and the transmission rate-ratio are known, and we compute the effective sharing interval in Section 3.3.7. The use of a circular buffer at the proxy allows us to share a single server stream (of the same movie) even over unicast channels, avoiding the need for costly multicast facilities. It also enables the proxy to start the suffix transmission after finishing the prefix transmission to a later-joining client, i.e., one channel between the proxy and a client is always sufficient, which represents a considerable cost saving in many cases. With this scheme, a client does not need a very large buffer, since it does not need to buffer the suffix data while playing the prefix data.

3.3.4 The Prefix Buffer Size

In this section, we compute the size of the prefix buffer, which depends on network latency, jitter, and traffic bursts. Let us assume that a proxy needs to pre-fetch T_{prefix} seconds equivalent of initial data of a video in a prefix buffer. Let the maximum and the minimum *server-to-proxy network latency* be $d_{s,p}^{\max}$ and $d_{s,p}^{\min}$ seconds respectively. Let the maximum *server-to-proxy jitter* be $\Delta_{s,p}^{\max}$, i.e., $\Delta_{s,p}^{\max} = d_{s,p}^{\max} - d_{s,p}^{\min}$. To overcome network latency and jitter suffered in a round trip, the proxy has to pre-fetch an equivalent amount of video data into the prefix buffer. For this reason, the size of the prefix buffer at the proxy needs to be

$$T_{prefix} = 2d_{s,p}^{\min} + 2\Delta_{s,p}^{\max} = 2d_{s,p}^{\max} \tag{3-1}$$

We use the same prefix buffer to assist in work-ahead smoothing of traffic bursts of a video. For this reason, we need to pre-fetch additional data in the prefix buffer. We call this additional data *burst-work-ahead data.* In our CBR-transmission scheme, the server streams the suffix data at a constant bit rate close to the mean bit rate, b bits/second, of the video. Let the server-proxy transmission rate-ratio be $r_{s,p}$. We can either pre-set a rate-ratio and compute the size of the burst-work-ahead data using Lemma 1, or pre-set a size for the burst-work-ahead data that a limited size proxy buffer can accommodate, and compute the rate-ratio using Lemma 2.

Let $r_{s,p}$ be fixed and let the size of the burst-work-ahead data be D_{burst}. From Lemma-1 we get expression (3-2).

$$\forall j\left[\left(j \in (1,\ldots,n)\right) \Lambda \left(\left(\sum_{k=1}^{j} f_k\right) \le \left(j \times r_{s,p} \times f_{mean} + D_{burst}\right)\right)\right] \tag{3-2}$$

If we pre-fetch D_{burst} bits of video data in the prefix buffer, the server's CBR suffix transmission of a VBR-encoded video will not cause any shortfall of video data at the

proxy. Dividing the burst-work-ahead data D_{burst} by $(r_{s,p} \times b)$ we can compute its equivalent time duration. We call this time duration *burst-work-ahead time* T_{burst}, i.e.,

$$T_{burst} = \frac{D_{burst}}{r_{s,p} \times b} \tag{3-3}$$

Adding these T_{burst} seconds of burst-work-ahead time, the size of our prefix buffer will be

$$T_{prefix} = (2d_{s,p}^{\max} + T_{burst}) \tag{3-4}$$

3.3.5 The Start-up Delay and The Playback Buffer Size

In this section, we compute the playback buffer size as well as the start-up delay of a video. We also use our CBR-transmission scheme (our work-ahead smoothing technique) to enable a proxy to stream the video data at a constant bit rate with reduced peak rate. We cannot, however, pre-fetch the data from a video in a client's buffer due to the uncertainty about a particular client's interest in a particular video. In order to perform work-ahead smoothing for a VBR-encoded video, a client needs to accumulate a certain amount of data in the playback buffer before starting the playback. We call this data *client-work-ahead* data and the delay *start-up delay*. Let $r_{p,c}$ be the rate-ratio of proxy-client transmission and D_{cw} be the size of the client-work-ahead data. From Lemma 1 we get expression (3-5).

$$\forall j \left[j \in (1..n) \wedge \left(\left(\sum_{k=1}^{j} f_k \right) \leq \left(j \times r_{p,c} \times f_{mean} + D_{cw} \right) \right) \right] \tag{3-5}$$

Now, if we assume the size of the playback buffer at the client is $T_{playback}$ seconds then

$$T_{playback} = \frac{D_{cw}}{r_{p,c} \times b} \tag{3-6}$$

We need to add the round-trip delay between the proxy and the client to $T_{playback}$ in order to compute the start-up delay. DSL and PON support jitter free transmission from the

proxy to the clients and the jitter in the Cable network is also negligible. For this reason, we assume that the *proxy-to-client network latency* $d_{p,c}$ has negligible jitter, i.e., $d_{p,c}$ is constant.

Let the start-up delay be d_s seconds. Therefore,

$$d_s = 2d_{p,c} + T_{playback} \tag{3-7}$$

We can reduce the start-up delay d_s to $2d_{p,c}$ seconds, and according to Lemma 2, we can reduce the size of D_{cw} to zero, thus increasing the proxy transmission rate, if the available bandwidths between the proxy and clients are great enough to support the increases.

There is a relation between the server transmission rate-ratio $r_{s,p}$ and the proxy transmission rate-ratio $r_{p,c}$. At the j^{th} frame time a proxy has only $\left(j \times r_{s,p} \times f_{mean} + D_{burst}\right)$ bits of cumulative video data to transmit to the client. Therefore, the proxy's cumulative transmission to any client cannot exceed this limit, i.e.,

$$\forall j\left[j \in (1..n) \wedge \left(j \times r_{p,c} \times f_{mean}\right) \leq \left(j \times r_{s,p} \times f_{mean} + D_{burst}\right)\right]$$

The above inequality converges into the following inequality:

$$\left(n \times r_{p,c} \times f_{mean}\right) \leq \left(n \times r_{s,p} \times f_{mean} + D_{burst}\right)$$

This sets an upper bound on the proxy to client transmission rate-ratio $r_{p,c}$ as expressed in expression (3-8).

$$r_{p,c} \leq \left(r_{s,p} + \frac{D_{burst}}{n \times f_{mean}}\right) = \left(r_{s,p} + \frac{D_{burst}}{M}\right) \tag{3-8}$$

Here, $M = n \times f_{mean}$ is the size of the entire video in bits. We assume the duration of the video is L seconds, i.e., the relation between L and M is $M = L \times b$.

3.3.6 The Smoothing Buffer Size for CBR-transmission (Work-ahead Smoothing)

In our CBR-transmission scheme, we use the smoothing buffer to do work-ahead smoothing in order to smooth out jitter and the traffic bursts. In our *scalable streaming* scheme, we use the smoothing buffer not only to smooth out jitter and traffic bursts but also to enable many clients to share the same server stream. Let the size of the smoothing buffer for a video in our scalable streaming scheme be $T_{smoothing}$ seconds. In Section 3.3.7, we compute the total size of smoothing buffer that is required for both CBR-transmission and scalable streaming. In this section, we compute only the part of the smoothing buffer that is required for work-ahead smoothing.

In Figure 3.5, the proxy receives a client request for a video at time 0. The proxy starts streaming prefix data to the client and requests suffix data from the server at the same time. The earliest possible time at which the proxy can receive the suffix data from the server is $2d_{s,p}^{\min}$. The proxy caches incoming suffix data in the smoothing buffer. The proxy starts overwriting cached suffix data in the smoothing buffer at time $(2d_{s,p}^{\min} + T_{smoothing})$. The proxy starts suffix data streaming from the smoothing buffer to the client at time T_{prefix}.

The proxy must not overwrite the cached data before transmitting it to the client, i.e.,

$$T_{prefix} \leq (2d_{s,p}^{\min} + T_{smoothing}) \qquad (3\text{-}9)$$

From expressions (3-4) and (3-9) we can derive

$$T_{smoothing} \geq \left(2\Delta_{s,p}^{\max} + T_{burst}\right) \qquad (3\text{-}10)$$

Here, the first term represents the amount of smoothing buffer space required due to network jitter and the second term represents the amount required due to encoding bit rate variation present in a VBR-encoded video. Therefore, the right hand side of

expression (3-10) is the minimum smoothing buffer size for each client when clients do not share a server-proxy stream or a smoothing buffer. This is used only for work-ahead smoothing in order to smooth out jitter and traffic bursts.

We have defined this as *work-ahead smoothing* buffer $T_{workahead}$, i.e.,

$$T_{workahead} = \left(2\Delta_{s,p}^{\max} + T_{burst}\right) \tag{3-11}$$

Work-ahead smoothing buffer space does not contribute to server-proxy stream sharing.

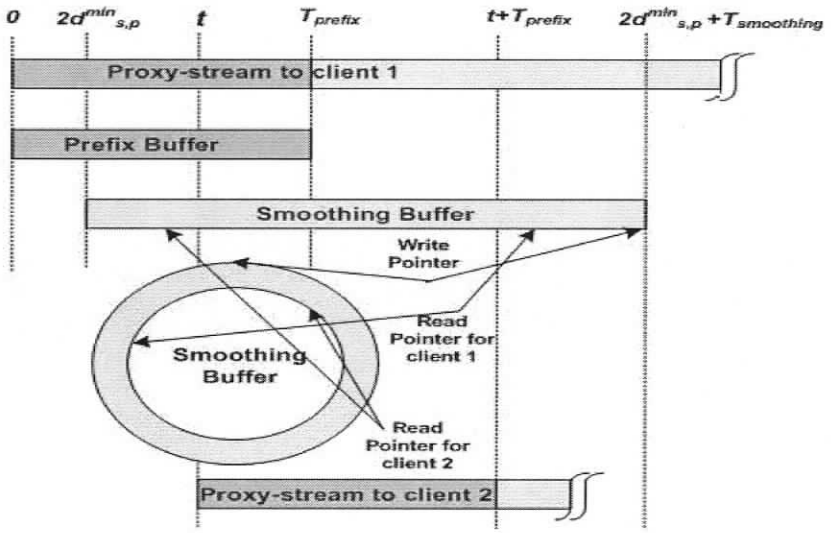

Figure 3.5 Prefix and suffix transmission from the prefix and the smoothing buffers at the proxy

3.3.7 The Size of the Smoothing Buffer for Scalable Streaming

In Section 3.3.6, we have computed the work-ahead smoothing buffer size, which is required to smooth out network jitter and traffic bursts for an individual client. When a group of clients share a server-proxy stream, i.e., a smoothing buffer, they can also share a common work-ahead smoothing buffer. The size of the common work-ahead smoothing buffer for a group of sharing clients remains the same as for an individual client and can

be computed using expression (3-10). In this section, we compute the size of the smoothing buffer, which is required to share the same server-proxy stream among many clients.

In Figure 3.5, the proxy receives another request for the same video from client 2 at time *t*. The proxy starts the prefix streaming to client 2 at time *t*. It wants to stream suffix data from the smoothing buffer that was allocated for this video after getting client 1's request. The proxy needs to start suffix streaming to client 2 at time $(t + T_{prefix})$. To guarantee that the proxy does not overwrite the suffix data before it has been transmitted to client 2, the following relation must hold.

$$(t + T_{prefix}) \leq (2d_{s,p}^{\min} + T_{smoothing}) \tag{3-12}$$

From expressions (3-4), (3-11) and (3-12) we have

$$t \leq \left(T_{smoothing} - T_{workahead}\right) \tag{3-13}$$

Expression (3-13) implies that when a smoothing buffer is allocated for a video, all the requests within the next interval t can share the same server-side suffix transmission as well as the same smoothing buffer, where the interval t is equal to the margin between the allocated smoothing buffer and the work-ahead smoothing buffer. If the margin between the allocated smoothing buffer and the work-ahead smoothing buffer is zero, then the clients cannot share the same server-proxy stream or the same smoothing buffer irrespective of the time differences between consecutive client requests. The larger the margin, the more requests can share the same smoothing buffer and the same suffix. For this reason, we recommend allocating as large as possible a smoothing buffer to a video, to make the margin large. We define this margin as the *sharing-smoothing-buffer* since it is used to enable many clients to share the same server-proxy stream.

We can pre-set the size of the sharing-smoothing-buffer in order to guarantee suffix and smoothing buffer sharing. Let the pre-set size of the sharing-smoothing-buffer be $T_{sharing}$. Then from expression (3-13) the size of the smoothing buffer will be

$$T_{smoothing} = (T_{sharing} + T_{workahead}) \tag{3-14}$$

The size of the smoothing buffer bounds the proxy transmission rate-ratio $r_{p,c}$ with the server transmission rate-ratio $r_{s,p}$. The difference between the amounts of video data received and transmitted at the proxy cannot exceed the capacity of the smoothing buffer, i.e.,

$$\forall j\left[j \in (1..n) \wedge \left(j \times r_{s,p} \times f_{mean} - j \times r_{p,c} \times f_{mean}\right) \leq T_{smoothing} \times r_{s,p} \times b\right]$$

$$\text{or} \quad \forall j\left[j \in (1..n) \wedge (r_{s,p} - r_{p,c}) \leq \frac{T_{smoothing} \times r_{s,p} \times b}{j \times f_{mean}}\right]$$

The above inequality yields:

$$\left(r_{s,p} - r_{p,c}\right) \leq \frac{T_{smoothing} \times r_{s,p} \times b}{n \times f_{mean}}$$

Therefore,

$$\frac{r_{p,c}}{r_{s,p}} \geq \left(1 - \frac{T_{smoothing} \times b}{M}\right) \tag{3-15}$$

Expression (3-15) sets a lower limit on the proxy to client transmission rate-ratio $r_{p,c}$.

3.3.8 The Average Streaming Cost

In this section, we compute the average streaming cost per client for a video. We consider both transmission costs and memory costs in computing the streaming cost. Two transmission costs are incurred in our streaming scheme; one from the server to the proxy and the other one from the proxy to the clients. Memory is required for the prefix and the

smoothing buffers at the proxy and for the playback buffers at the clients. The smoothing buffer space is dynamically allocated and freed by a proxy based on demand. The proxy allocates a smoothing buffer to a video if it gets a new request and there is no previously allocated smoothing buffer to share. If another request for the same video comes within the next interval of $T_{i,sharing}$ seconds it shares the previously allocated smoothing buffer. Where $T_{i,sharing}$ is the sharing interval set a priori for video *i*. The proxy may allocate more than one smoothing buffers for the same video if the time differences between two consecutive requests are more than $T_{i,sharing}$ seconds. We assume that the arrival of the requests of a video is a Poisson process and λ_i is the arrival rate of the requests of video *i*. According to Poisson probability distribution the expected number of requests for video *i* in the interval $[0, T_{i,sharing}]$ is $\lambda_i T_{i,sharing}$. Therefore, the total number of requests that share a single smoothing buffer or a single server stream of video *i* is $\lambda_i T_{i,sharing}$ requests plus the one request that occurs just before the interval starts and triggers a smoothing buffer allocation event.

We assume that c_{st} is the cost of transmitting one bit of video data from the server to the proxy, c_{pt} is the cost of transmitting one bit of video data from the proxy to the client, and c_m is the cost of memory to buffer one bit of video data. We can say that $(1 + \lambda_i T_{i,sharing})$ clients share $(L_i - T_{i,prefix} \times r_{s,p}) b_i$ bits of suffix transmission and $T_{i,smoothing} \times r_{s,p} \times b_i$ bits smoothing buffer. Here, L_i is the duration of video i in seconds. Therefore, the suffix transmission cost and smoothing buffer cost per client are $\frac{c_{st}(L_i - T_{i,prefix} \times r_{s,p}) b_i}{1 + \lambda_i T_{i,sharing}}$ and $\frac{c_m \times T_{i,smoothing} \times r_{s,p} \times b_i}{1 + \lambda_i T_{i,sharing}}$ respectively. Since there are λ_i requests in the mean for video *i,* we can compute the total suffix transmission cost and total smoothing buffer cost for the video by multiplying the respective per client cost by λ_i. All the clients of video *i* share the prefix buffer of size $T_{i,prefix} \times r_{s,p} \times b_i$ bits, i.e., the total prefix buffer cost for the video

is $c_m \times T_{i,prefix} \times r_{s,p} \times b_i$ and is independent of the number of clients for the video. Finally, adding all of the relevant costs and substituting $T_{i,sharing}$ using expression (3-14), we get the total cost (transmission plus buffering) C_i for video i as follows:

$$C_i = c_{st} \frac{L_i - T_{i,prefix} \times r_{s,p}}{1 + \lambda_i \left(T_{i,smoothing} - T_{i,workahead}\right)} b_i \lambda_i + c_{pl} L_i b_i \lambda_i + c_m \frac{T_{i,smoothing} \times r_{s,p}}{1 + \lambda_i \left(T_{i,smoothing} - T_{i,workahead}\right)} b_i \lambda_i$$

$$+ c_m \left(T_{i,prefix}\right) r_{s,p} \times b_i + c_m T_{playback} \times r_{p,c} \times b_i \lambda_i \qquad (3\text{-}16)$$

Alternatively,

total cost = *suffix transmission cost* from the server to the proxy + *prefix plus suffix transmission cost* from the proxy to the clients + *smoothing buffer cost* + *prefix buffer cost* + *playback buffer cost*.

Since there are λ_i requests on average for video i, dividing C_i by λ_i we can find the average streaming cost per client, $C_{i,average}$, for video i as follows:

$$C_{i,average} = \frac{C_i}{\lambda_i} \qquad (3\text{-}17)$$

3.4. Simulation Results

We have written a Java simulation program to analyze the performance of our CBR-transmission and scalable streaming schemes. We have used the video trace of a high quality MPEG-4 encoded movie, namely Jurassic Park I [80]. The length of this video is 1 hour, i.e., there are 90000 frames at a frame rate of 25 frames/second. The mean and the peak bit rate of the video are 0.77 Mbps and 3.3 Mbps respectively. The standard deviation of frame size is 2259.05 bytes. We assume the maximum total (transmission + propagation + queuing) delay between the server and the proxy is 500ms, the minimum

total delay between the server and the proxy is 100ms, and the delay between the proxy and a client is 100ms.

We have assumed the transmission rate equal to the mean bit rate 0.77 Mbps, i.e., $r_{s,p} = 1$ and T_{burst}=161.66 seconds. We have used $r_{p,c} = r_{s,p} = 1$, which complies with the upper limit (3-8) and the lower limit (3-15) of the proxy transmission rate-ratio, $r_{p,c}$. Due to the above choice of $r_{p,c}$, we got $T_{playback} = T_{burst} = 161.66$ seconds. According to expressions (3-4) and (3-11), the sizes of our T_{prefix} and $T_{workahead}$ are 162.66 seconds and 162.46 seconds respectively. We have used 3 minutes or 180 seconds of smoothing buffer, which is equivalent to 17.325 MB of buffer space. With the above settings the size of $T_{sharing}$ becomes 17.54 seconds.

Figure 3.6 plots the prefix buffer and the smoothing buffer requirements against the rate-ratio. In this part of our simulation, we have chosen a small sharing buffer (17.54 seconds or 1.69 MB). For this reason, the difference between the smoothing buffer and the prefix buffer becomes only 1.69 MB and their graphs coincide with each other. The total buffer requirement decreases as the rate-ratio increases. This is because the server streams the video data at higher rates for higher rate-ratios, which decreases the buffer amounts required for work-ahead-pre-fetch data both in the prefix buffer and in the smoothing buffer. Figure 3.6 shows the trade-off between the buffer space and the bandwidth.

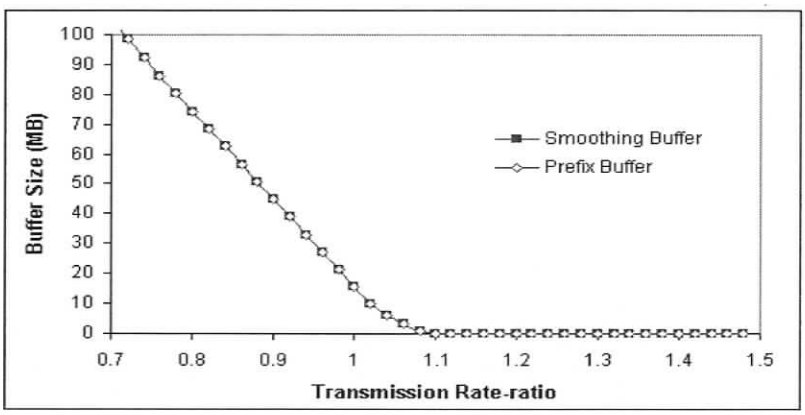

Figure 3.6: Buffers (Prefix, Smoothing) vs. Rate-ratio

Figure 3.7 plots the cumulative data of CBR transmission and CBR reception of the video Jurassic Park I. Here, the cumulative video data at a particular media-time is the summation of the bits in the video frames from the beginning of the video up to the media-time. The cumulative transmitted or received data at a particular media-time is the summation of the bits transmitted or received from the beginning of the transmission or reception, respectively, up to the media-time. The top graph, which is for the summation of the prefix and the cumulative suffix data that could possibly be available at the proxy if there were no network latency and the server were transmitting the video data at variable rates, sets the upper limit for the cumulative data that can be available at the proxy. The bottom graph, which is for the real-time cumulative data required at the client, sets the lower limit for the cumulative data that the client must receive in order to get hiccup free playback. The server transmits the video at a constant bit rate, which is 0.77 Mbps in our experiment. From Figure 3.7 we observe that the available cumulative data at the proxy never crosses the upper limit set by the top graph. The cumulative data that is available at the proxy also sets the upper limit for the cumulative data that can be transmitted from the proxy. Like the server, the proxy transmits the video at a constant bit rate, 0.77 Mbps. Figure 3.7 shows that the proxy cumulative data transmission never crosses the upper limit set by the available data at the proxy. Figure 3.7 also shows that

the client cumulative data reception never falls below the lower limit set by the bottom graph.

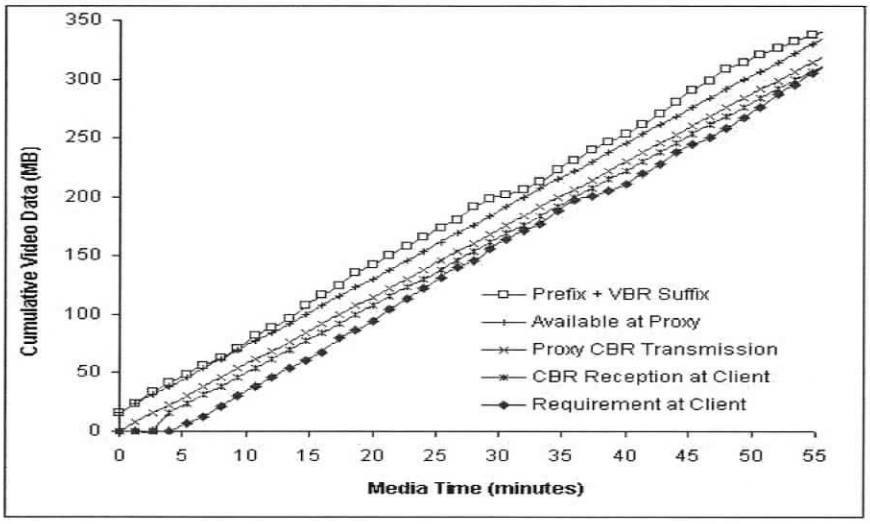

Figure 3.7: Cumulative Data vs. Media Time

For the rest of our experiment, each time we run our simulation program, we assume that 100 videos are being served through the proxy and the aggregate access rate for these videos is $\lambda = 600$ requests/minute. We assume that the different videos in the same run have varying popularities, i.e., different access rates. We have ranked the videos according to their popularity. Bestavros et al. [76] observed that the distribution of web requests follows a Zipf-like distribution: $\rho = \frac{k}{i^{\alpha}}$, where ρ is the probability that web object i will be accessed and $k = \frac{1}{\sum_{i=1}^{N}\left(\frac{1}{i^{\alpha}}\right)}$ when N web objects are ranked according to their popularity. The parameter $0.0 \le \alpha \le 1.0$ is the sensitivity parameter of the distribution. If a value close to zero is used for α the access probability will be less sensitive to the popularity. On the other hand, if a value close to one is used for α the access probability will be more sensitive to the popularity. We have used the same distribution for the videos. We draw the access probability ρ_i for video i from a Zipf-like

distribution. We use $\alpha = 0.3$ in order to make the access probability of the videos not too sensitive to their popularity. Multiplying the aggregate access rate λ by ρ_i we derive the access rate λ_i for video i. In order to show that our transmission scheme performs the same with various choices of the costs of c_{st}, c_{pt}, and c_m we have run our simulation with various costs ratios of $c_{st} : c_{pt} : c_m$.

The smoothing buffer is used to facilitate the resource sharing in our UniSMerge transmission scheme. SBatch scheme uses the prefix buffer to do the same. For this reason, the average costs against different smoothing buffer lengths in UniSMerge are analogous to the average costs against different prefix buffer lengths in SBatch. We have computed the average streaming cost in our UniSMerge transmission scheme using equation (3-17). Figure 3.8 plots the normalized average streaming cost per client against the smoothing buffer length under UniSMerge and Online Smoothing [17] transmission schemes and the average cost against the prefix buffer length under SBatch [18] transmission scheme. The simulation result shows that the highest average streaming cost occurs in SBatch transmission scheme when its prefix buffer size is zero. We have normalized all the average streaming costs in all the transmission schemes with respect to this highest cost value. We have used the cost ratio $c_{st} : c_{pt} : c_m = 2:1:1$ for Figure 3.8. It shows that the average cost always remains very high and increases with an increase in smoothing buffer length in Online Smoothing. This is because an increase in buffer length increases the cost but this increased buffer does not facilitate resource sharing among more clients. The graph of SBatch cost shows that the average cost decreases very sharply at the beginning as the length of the prefix buffer increases. But after a certain point the cost starts to increase and it continues. However, the average cost in our UniSMerge transmission scheme is always decreasing with an increase in the length of the smoothing buffer. It also decreases very sharply at the beginning. So, we can always add more smoothing buffer in our transmission scheme to reduce the cost. Figure 3.8 also shows that the cost in our UniSMerge transmission scheme is the lowest most of the time compared to that of other transmission schemes.

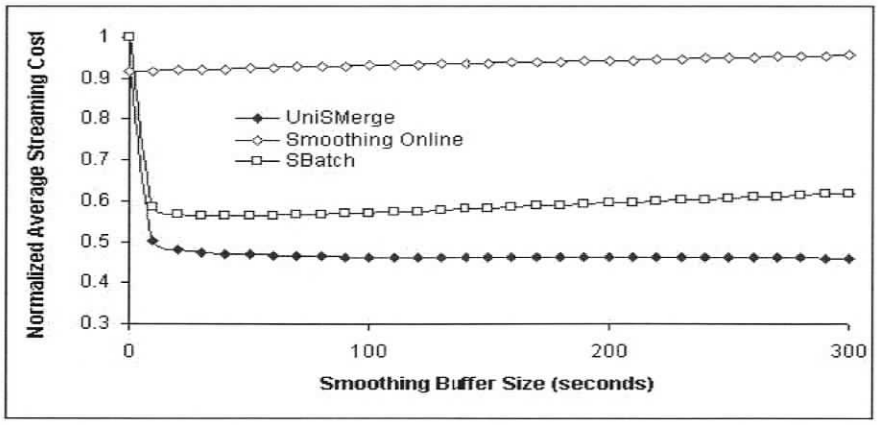

Figure 3.8: Normalized Average Streaming Cost vs. Smoothing Buffer Length

Figure 3.9 plots the normalized average streaming cost per client against the aggregate access rate for the videos under UniSMerge, Online Smoothing [17], and SBatch [18]. We have also used the cost ratio $c_{st} : c_{pt} : c_m = 2:1:1$ for Figure 3.9. Our simulation result shows that the highest average streaming cost occurs in Online Smoothing transmission scheme when the aggregate request rate is 1000 requests/sec. We have normalized all the average streaming costs in all the transmission schemes with respect to this highest cost value. The cost remains highest and constant with the request rate in Online Smoothing, since, it is not sharing either suffix transmission from the server or the smoothing buffers at the proxy. Like SBatch our UniSMerge transmission scheme yields lower cost when the request rate increases. However, UniSMerge cost is always less than that of SBatch, and Online Smoothing. When the aggregate access rate increases, more clients are sharing the resources such as the suffix data, the prefix and the smoothing buffers in our UniSMerge transmission scheme, i.e., it scales the streaming service more economically.

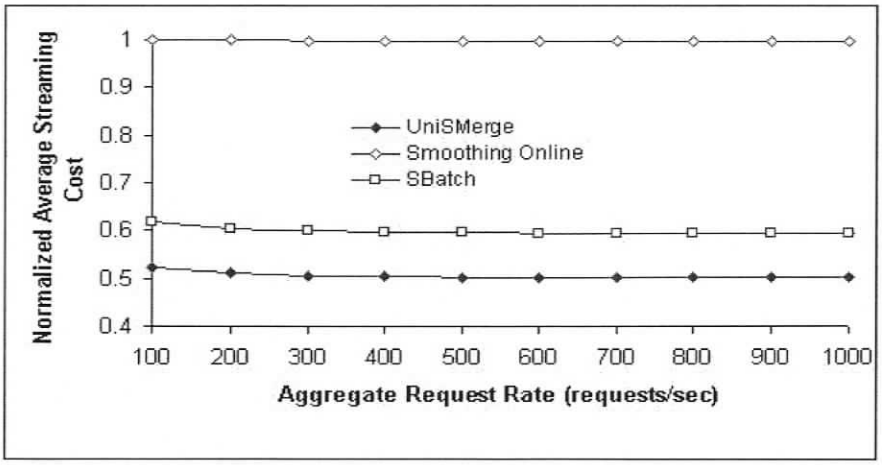

Figure 3.9: Normalized Average Streaming Cost vs. Aggregate Request Rate

Figure 3.10 plots the normalized average streaming cost per client against burst-work-ahead time T_{burst}. The simulation result shows that the highest average streaming costs occur in our UniSMerge transmission scheme when the costs ratio 1:1:1 is used. We have normalized all the average streaming costs with respect to the highest cost values in Figure 3.10, Figure 3.11, and Figure 3.12. The length of prefix buffer T_{prefix} as well as the length of work-ahead smoothing buffer $T_{workahead}$ increases as the value of T_{burst} increases. The window size of the smoothing buffer $T_{smoothing}$ is fixed at 180 seconds. The length of the work-ahead smoothing buffer $T_{workahead}$ increases as the length of T_{burst} increases. The length of $T_{sharing}$ decreases as the length of the work-ahead smoothing buffer $T_{workahead}$ increases. As the length of $T_{sharing}$ decreases fewer clients share the same server stream and the same smoothing buffer. As resource sharing decreases the average streaming cost increases. For this reason the average streaming cost increases as T_{burst} time increases in Figure 3.10. The length of T_{burst} for a video is higher if the video has higher traffic bursts. This means the average streaming cost of a video with higher traffic bursts will be higher than the average streaming cost of a video with lower traffic bursts using the same size smoothing buffer. In Figure 3.12, we will see that we can reduce the average streaming

cost of a particular video by increasing its smoothing buffer size. In Figure 3.10, there is a large jump in average cost when T_{burst} is 179.2 seconds. At this point, the length of $T_{workahead}$ is almost equal to the length of $T_{smoothing}$ and the length of $T_{sharing}$ is zero. No client shares the same server stream and the same smoothing buffer with other client; therefore, the average streaming cost rises sharply.

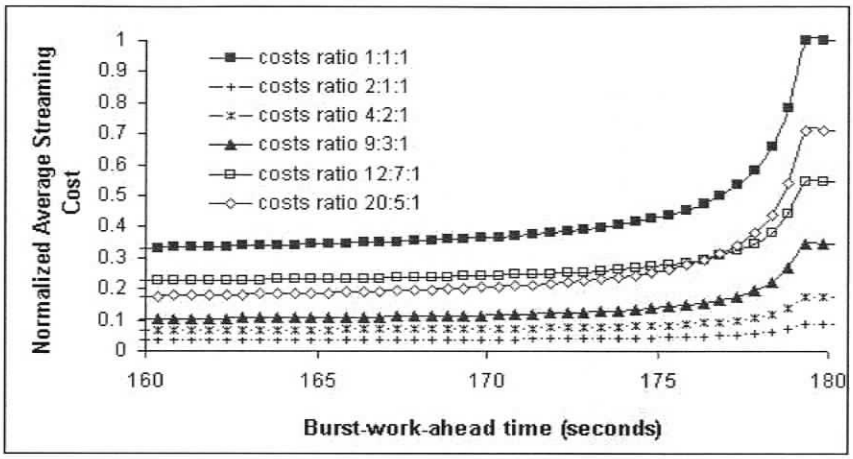

Figure 3.10 Normalized Average Streaming Cost vs. Burst-work-ahead time

We have used T_{burst}=161.66 seconds in order to keep it less than the critical size 179.2 seconds, in the rest of our experiment.

Figure 3.11 plots the normalized average streaming cost against the aggregate request rate for the videos. It shows that our transmission scheme yields lower average streaming cost when the request rate increases. This is because as the request rate increases more clients are sharing the prefix, server streams, and smoothing buffers at the proxy. The increased degree of resource sharing overpowers the increases in resource requirement at the higher request rates at the proxy, which enables our streaming scheme to scale a streaming service economically.

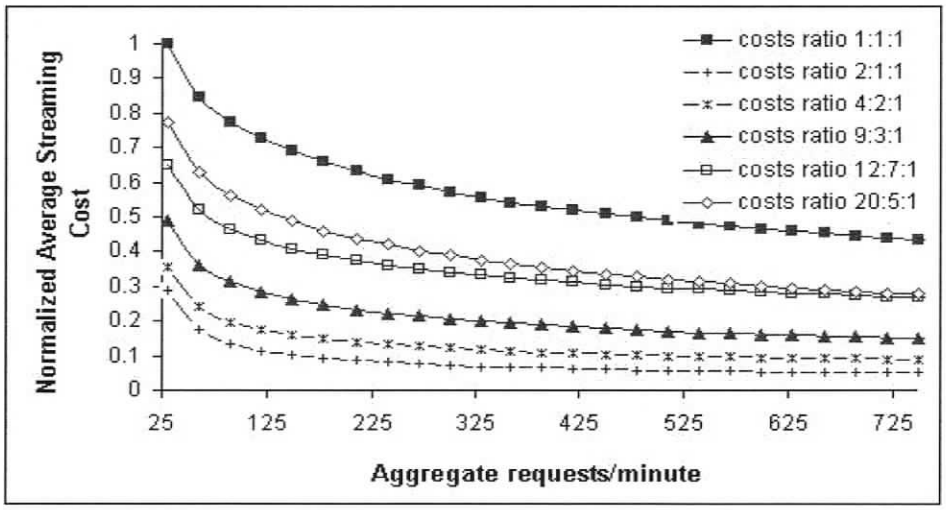

Figure 3.11 Normalized Average Streaming Cost vs. Aggregate Request Rate

Figure 3.12 plots the normalized average streaming cost against the window size of the smoothing buffer $T_{smoothing}$. This graph shows that the average streaming cost decreases very sharply when the size of $T_{smoothing}$ starts to offset the size of $T_{workahead}$, i.e., at 162.46 seconds. After a sharp fall, the average streaming cost also continues to decrease as the increase in the size of $T_{smoothing}$ increases the size of $T_{sharing}$. A larger size of $T_{sharing}$ allows more clients to share a single server stream and a single smoothing buffer at the proxy, i.e., reduces the streaming cost per client. This demonstrates that by adding more smoothing buffer we can reduce the average streaming cost in our transmission scheme.

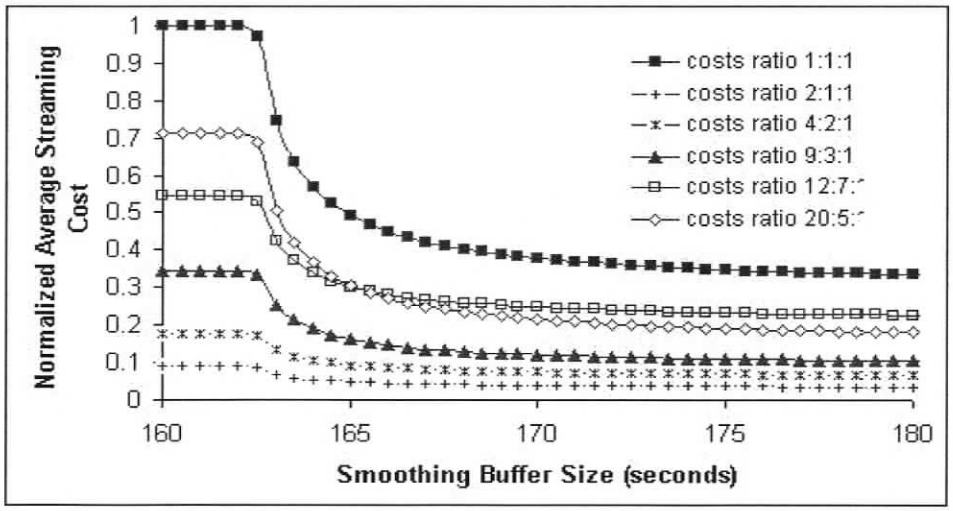

Figure 3.12 Normalized Average Streaming Cost vs. Smoothing Buffer Size

3.5. Chapter Summary

We have presented a proxy-based scalable streaming scheme in this chapter. In our streaming scheme, a proxy enables many clients to share a single server stream, a prefix buffer, and a smoothing buffer. This makes our streaming scheme highly scalable, increases the overall throughput, and reduces the load on the server, keeping the average cost low. It scales the streaming service economically, i.e., as the number of clients grows the cost per client goes down. A proxy overcomes the network latency problem by storing the prefix ahead of time, and by streaming the prefix immediately after intercepting the client's request. A proxy absorbs the network jitter by doing work-ahead smoothing. For this reason, the clients get hiccup free media playback. It enables suffix sharing using a small buffer and without using multiple channels at each client. Our proxy always uses a work-ahead-smoothing buffer. This eliminates the requirement of just-in-time arrival of video data at the proxy. The smoothing buffer also enables our scheme to transmit VBR encoded videos smoothly and to share the same server-proxy stream among many clients. Work-ahead smoothing by prefix and smoothing buffers eliminates the requirement of

either using large peak bandwidth or using a complicated online smoothing technique to transmit VBR-encoded video.

Interactive playback is an important feature that a streaming solution should provide. The standard frame sequence of an MPEG encoded video stream is suitable for normal playback. However, its inter-frame dependency makes it difficult to use for interactive playback modes, such as play backward, jump or fast forward/backward. A scalable streaming system is required to serve videos to a large number of clients and it needs to share a single server stream among many clients. However, different interactive clients generally ask for different playback sequences. Hence, their streams cannot be grouped and served together by a single server stream. Therefore, as the frequency of interaction increases, an ordinary scalable streaming service degenerates into a non-scalable service. In the next chapter, we present an interactive and scalable streaming scheme that resolves the above problems efficiently.

4. An Interactive and Scalable Multimedia Streaming Scheme

4.1. Introduction

Multimedia streaming applications, such as video-on-demand and videoconferencing, need to provide *interactive* playback as well as normal playback for users. The commonly required modes of interactive playback are *play backward (PB), pause (PA), jump forward (JF), jump backward (JB), fast forward (FF), fast backward (FB), slow forward (SF) and slow backward (SB)*. Under normal playback, a video is played at a normal rate and in the forward direction. In play backward mode, a media is also played at a normal rate but in the reverse direction. When a media is paused its playback is suspended for a period, called paused period. At the end of a paused period normal media playback is resumed from the point where it was paused. A jump forward playback moves the playback point of a media to an arbitrary location ahead of the current playback point. Similarly, a jump backward playback moves the playback point to an arbitrary location behind the current playback point. In the fast forward playback, the media is played in the forward direction and at a higher rate than that of the normal playback rate. In the fast backward playback, the media is also played at a higher rate than that of the normal playback rate but in the reverse direction. Slow forward playback plays the media in the forward direction but at a rate slower than the normal playback rate. Similarly, slow backward playback plays the media in the reverse direction and at a rate slower than the normal playback rate.

Large video files are usually compressed using encoders, such as MPEG-2 [12][13], MPEG-4 [14][15]], and H.261 [10]. These video encoders generally create three types of compressed frames: I-frames, P-frames, and B-frames. The Macro-blocks of a P-frame are predicted from a previous I-/P-frame. A B-frame is predicted from both the previous I/P-frame and the next I/P-frame. A typical frame sequence of a compressed video stream

is I-B-B-P-B-B-P-B-B-P-B-B-I-B-B-P-, which is only suitable for normal playback. Motion compensated forward predictive coding makes it difficult to play the above sequence in an interactive playback mode.

A scalable streaming scheme is useful to serve large numbers of concurrent clients. There are two types of scalable streaming schemes. One type uses a multicast channel to serve many clients by a single stream multicast from the server [91]. This scheme needs multicast capable networks, which is not widely available at this time. The other type of scalable scheme uses a proxy between the server and the clients in order to enable a single server-proxy stream sharing among a group of clients. This type of scalable scheme does not need multicast capable networks; it works fine with unicast capable networks, which is widely available now. We have proposed a proxy based scalable streaming model in Chapter 3 of this type. Both types of scalable streaming schemes share a single server stream among a group of clients. An interactive client, however, needs a unique playback sequence; hence, interactive clients cannot be served together by a single server stream. Thus as the number of interactive clients rises, an ordinary scalable streaming services tend to lose scalability.

In this chapter, we propose an improved proxy-based scalable streaming scheme suitable for use over the Internet. We use video segmentation and hybrid temporal-data-partition scalable video encoding to create a suitable playback sequence for interactive playback modes. A video is encoded into two sub-streams: base and enhancement sub-streams. The base sub-stream, which is a low quality moderate size sub-stream, is cached at the proxy in order to play it during FF/FB playback. A video is logically divided into several segments. Each segment is then divided into a prefix and a suffix. All the segment prefixes are cached at the proxy in order to allow an interactive client to jump forward/backward on those prefixes. Smoothing buffers at the proxy temporarily buffer the suffix data transmitted from the server in order to provide with PB, PA, SF, and SB interactive playback modes. We use the same prefix and smoothing buffers at the proxy to overcome network jitter, the traffic burst problem of variable-bit-rate (VBR)-encoded

videos, and to share a server-proxy stream among many clients. The main benefits of our scheme are as follows:

- It is scalable, uses proxy instead of multicast channel and supports the entire generally required interactive playback modes from the proxy without losing scalability.

- The proxy without any server intervention supports all the interactive playback modes.

- It uses only one stream composed of two sub-streams and does not need dedicated interactive server stream to support client interactions.

- It does not need a large client buffer and complicated processing at each client.

- It does not increase server-proxy bandwidth and the proxy buffer requirements as the level of interaction increases and is able to serve any large number of clients using certain part of the available server-proxy network bandwidth and proxy buffers.

The rest of this chapter is organized as follows. In Section 4.2, related research works are presented. In Section 4.3, we describe our proxy-based interactive-scalable streaming scheme. We discuss the segmentation and scalable encoding techniques, which enables us to provide interactive playback modes, in Section 4.4. In Section 4.5, we describe different transmission schemes, necessary to support different interactive playback modes. In Section 4.6, we compute the sizes of the base, prefix, and smoothing buffers required at the proxy. We also compute the amount of server-proxy network bandwidth required in our streaming model in Section 4.6. We present some experimental results in Section 4.7. Finally, Section 4.8 concludes this chapter.

4.2. Related Research Works

Lin et al. [23] have mentioned a straightforward implementation of backward playback that needs a decoder to decode the whole Group of Picture (GOP), store all the decoded frames in a buffer, and play decoded frames backward. Unfortunately, this technique needs a large buffer at the client. Storing all the decoded frames, i.e., nearly the whole video in the client is not desirable either. Lin et al. [23] have also mentioned an alternative straightforward implementation of backward playback. In this alternative implementation, a client decoder decodes the GOP up to the frame that needs to be played, and then goes back to decode GOP again up to the next frame that needs to be played next and so on. This implementation requires the client machine to operate in an extremely high speed. Lin et al. [23] have proposed dual bit-streams: a forward encoded stream for forward playback and a special reverse encoded stream for reverse playback. They have kept I-frames in the reverse bit-stream interleaved between the I-frames in the forward bit-stream. The server can switch from one bit-stream to another bit-stream. In their scheme, I-frames represent the point of access to decode the sequence from any arbitrary position. When the client requests the backward playback the reverse encoded bit-stream is used. I-or P-frames of one bit-stream are often used to approximate P-frames of the other bit-stream while switching the playback mode. This approximation may lead to frame mismatch and cause drift when the approximated frames are used as the reference frames to predict the following P/B-frames. And the drift lasts until the next I-frame. Two more bit-streams consisting of all P-frames are used to compensate for the drift during bit-stream switching. One drift-compensation bit-stream D^{FR} is used for switching from I- or P-frames of the forward bit-stream F to the P-frames of the reverse bit-stream R and the other one D^{RF} is used for switching from I- or P-frames of R to the P-frames of F. Adding too many bit-streams requires more disk space at the server, which makes the schemes proposed in [23] less useful.

A jump action may bring the play point to an arbitrary P or B frame. All the referred P/I frames from this frame need to be transmitted through the network and decoded by the decoder. For this reason, the network will need to transport frames at a much higher rate and the decoder will also need to decode frames at a much higher rate. Alternatively random access points can be fixed and limited only to I-frames. None of the above is a good solution with respect to transmission bandwidth and decoder complexity. Lin et al. [23] have also proposed a least cost frame selection scheme from their dual bit-streams in order to reduce decoding complexity and network traffic in random access mode under a jump action. However, the usage of multiple bit-streams makes their scheme less attractive.

A straightforward implementation of fast forward playback at a speed-up factor of x is to transmit and decode x times faster than the normal playback has been proposed in [88][28]. Unfortunately, this approach increases server load, bandwidth requirement, and decoder speed to a large extent. An alternative approach has been proposed in [89]. In [89], some frames are skipped and only x^{th} frame are transmitted, decoded and played in order to keep bandwidth requirement and decoder speed at an acceptable limit. The main problem of this technique is that a decoder could not decode some selected frames due to the fact that their reference frames have been skipped. In order to avoid the above problem, some segments are skipped instead of some frames and every x^{th} segment is transmitted, decoded, and played in [26]. In this case, all the reference frames of a frame in a segment reside in the same segment and are available to decode the target frame. However, it needs special segment placement and segment selection schemes at the server. One segment placement and segment selection scheme can support only one fast forward speed. Noticeable visual discontinuities result from entire segment skipping that may not be acceptable. Reference [26] has proposed another alternative, which transmits and plays only the I-frames in the fast forward playback. In this technique, fast forward speed depends on GOP structure and is equal to the GOP size. Moreover, it will need very high bandwidth to transmit all the I-frames. Another alternative is to use separately

encoded low-resolution fast forward bit-stream in the fast forward playback, which has been proposed in [27][90]. Unfortunately, this technique requires multiple low-resolution fast forward bit-streams to support multiple fast forward speeds, i.e., requires very large disk space at the server to store multiple bit-streams, and playback suffers from drift when it switches normal stream to fast forward stream and fast forward stream to normal stream.

Gao et al. [25] proposed to use scalable encoding in order to create a base sub-stream and an enhancement sub-stream and use only the base sub-stream in the fast forward playback. They proposed to use both sub-streams in the normal playback. They have proposed their scheme only for a simple streaming model. Their scheme does not work with a scalable streaming scheme. Their proposed layered base soft real-time scheduling algorithm supports only fast forward playback. Other interactive playbacks, such as play backward, fast backward etc. remain unsupported. For these reasons, their scheme has very limited use.

Fast backward is the most difficult interactive playback to implement. It faces all the challenges faced by fast forward playback and backward playback together. Reference [26] has proposed a straightforward implementation that is to transmit and play only I-frames in the reverse order. Transmitting only the I-frames at a higher speed is more bandwidth consuming. Reference [27] has proposed to use separately encoded low-resolution fast backward stream in fast backward playback. Separately encoded bit-streams need a large storage at the server.

Feng et al. [28] proposed a buffer based scheme, where some of the most recently received frames are stored in a client buffer called VCR-window. All the backward VCR capabilities, such as FB, JB and SB, are fully functional within the VCR-window without requiring changes to the bandwidth reservation made between the server and the client. However, there are several limitations to this scheme, such as every client needs to

maintain a large buffer or a VCR-window. Though the backward playback capabilities are fully functional within the VCR-window, it cannot support any forward operation.

Liao et al. [29] proposed a Split and Merge (SAM) protocol to minimize the problem of supporting VCR functionalities in a shared streaming model. SAM starts by serving clients in a group. When a client in a group initiates a VCR interaction, the protocol splits off the interactive client from the original group and temporarily assigns a new video stream called interactive stream to that client. With a dedicated video stream the client can perform any VCR operation desired. As soon as the VCR operation terminates, SAM merges this client back to a nearest running group. The problem with the SAM protocol is that it needs many dedicated interactive streams when many clients request VCR operations, i.e., it loses scalability as interaction increases. Switching between different streams during split and merge also causes drifts or glitches in the playback. Different interactive playback modes need different types of playback sequences.

Though the interactive streaming schemes proposed in [23], [28], [25], and [29] look somewhat similar to that of ours, our proposed scheme outperforms them in many respects. Our scheme supports DVD like playbacks with scalable transmission scheme and uses only one stream, which consists of sub-streams to, support various DVD controls from the proxy. However, the scheme proposed in [23] does not work with scalable transmission scheme, uses many streams and huge server storage to store those streams, and supports user controls from the server instead of proxy. Therefore, the server becomes overloaded. These issues make our scheme favourable compared to that of [23]. Our scheme not only works with scalable transmission scheme but also supports the entire, generally required DVD like controls. On the other hand, the scheme in [25] does not work with scalable transmission scheme and does not support all the controls generally required. Further, the scheme proposed in [28] cannot support any kind of forward controls, such as fast forward, jump forward etc. and needs a large client buffer and complicated control processing at each client. Neither [25] nor [28] support user

controls from the proxy. SAM protocol in [29] has been proposed only for multicast-enabled networks. SAM uses a multicast stream to deliver video data to many clients during normal playback, however, it uses individual interactive stream (unicast) from the server for each client when the client is in VCR like operation. Unlike SAM, we have defined our scheme for unicast-enabled networks, where a proxy resides between the server and the clients in order to enable many clients to share the same server-proxy stream. Our scheme does not need to use individual interactive stream from the server for any client when it is in DVD operation. Our scheme has the mechanism to create and transmit different types of playback sequences required for different interactive playback modes.

4.3. Proposed Interactive Streaming Scheme

We believe that from a user's point of view, DVD like control is more suitable than a VCR like control for streaming digital media. We have also observed that a user usually skips a significant part or a segment of a video through an interactive playback. For this reason, we propose to provide segment-by-segment DVD-like interactive playback controls, where a user can jump forward or fast forward to the beginning of any segment next to its current segment; a user can jump backward or fast backward to the beginning of its current segment or the beginning of any segment previous to its current segment; a user can pause, slow forward or backward, and play backward in its current segment.

We have proposed a constant-bit-rate (CBR) streaming scheme for variable-bit-rate (VBR) encoded videos in Chapter 3. In Chapter 3 we have also proposed a scalable streaming scheme, where many clients can share a single server stream through a proxy, with CBR streaming of VBR-encoded videos. In this chapter, we are proposing a complementary scheme to support segment-by-segment DVD controls to the users who share a server stream. We propose to support DVD controls for a client by adapting the data flow of its dedicated proxy-client stream and leaving the shared server-proxy stream unchanged. Interactive clients remain in the shared mode before, during, and after the

interaction. If the interaction is either jump-forward or fast-forward, the client will join a new group. Otherwise, the client will remain in the original group.

We propose to use scalable encoding to encode the original video stream into base and enhancement sub-streams and to use the base sub-stream as the playback sequence during FF/FB playback. Encoding a video stream into base and enhancement sub-streams can be done either at the server or at the proxy. If the server encodes the original stream into the sub-streams then the base sub-stream is pre-fetched at the proxy. In this case, the server needs to transmit only the enhancement sub-stream, which saves a certain amount of server-proxy network bandwidth. The proxy adds the enhancement sub-stream with the base sub-stream to reproduce the original stream for normal playback. If the server does not encode the original stream into sub-streams, the proxy receives the original stream from the server, decodes it, and encodes it into base and enhancement sub-streams. The proxy caches the base sub-stream in order to use it as the FF/FB playback sequence. This mechanism liberates the server from scalable encoding. The proxy implements scalable encoding and interactive-scalable streaming scheme keeping the server transparent. However, this option needs relatively more server-proxy network bandwidth and adds more complex processing, such as decoding and encoding at the proxy.

We propose to divide the videos into several logical segments. The sizes of our segments might be different or equal. A segment may be comprised of a video sequence, a scene, or a shot. There are many mechanisms [94][95][96] to detect shots, scenes, and video sequences in a video. We propose to use these existing mechanisms to divide a video into many segments. We also propose to divide each segment into prefix and suffix. From the previous chapter, we know that the prefix of each segment must be pre-fetched at the proxy in order to perform work-ahead smoothing on the VBR data of the rest of the segment. Pre-fetching the prefixes of all segments allows clients to request and share the video, segment by segment. This also allows a client to jump forward/backward to the beginning of a segment.

When a client needs to play a particular segment, the proxy immediately transmits the prefix to it and sends a request for the suffix of the segment to the server. The server transmits the segment suffix to the proxy at a fixed rate, which might differ from the mean bit-rate of the video. The incoming segment suffix is buffered into a smoothing buffer at the proxy and transmitted to the client when prefix transmission is finished.

The proxy allocates a flat smoothing buffer, which is equal to the size of the segment suffix. If another client needs to play the same segment within a time interval equal to the segment length (expressed in seconds required to play it at normal speed,) it can share the same segment suffix and smoothing buffer at the proxy. This helps to make the streaming scheme highly scalable. Another client may need to play the same segment, as the next segment in the sequence of its normal, continuous playback or as the new segment required by an interaction.

The smoothing buffer remains allocated to a segment for as long as the clients require it, either for normal playback or for an interaction. The data in the segment's smoothing buffer is not overwritten. PB and SB interactions can use the residual data in the smoothing buffer and the pre-fetched data in the prefix buffers. Using the data in the buffers also supports PA and SF interactions. Conceptually our smoothing buffer for a segment works as the VCR-window of [28]. Unlike [28], we don't maintain a rewind buffer, and we implement the VCR-window at the proxy rather than at each client. Also, we share the window among many clients. A smoothing buffer must be de-allocated when no client requires it.

4.4. Segmentation and Scalable Encoding to support DVD Controls

Scalable encoding, such as signal-to-noise ratio (SNR) scalability, Spatial scalability, Temporal scalability, and Data partitioning have been defined for MPEG-2 and MPEG-4.

In these scalable encoding techniques, a single stream consists of multiple sub-streams, e.g., one base sub-stream and one or more enhancement sub-streams. And a subset of these sub-streams, which always includes the base sub-steam, is chosen under a constraint condition. These are called Coarse Granular Scalable (CGS) encoding techniques since it allows the inclusion and the exclusion of a whole sub-stream only. A Fine Granular Scalable (FGS) encoding has been proposed in [93] for MPEG-4, which allows the inclusion and the exclusion of a sub-stream as well as a part of a sub-stream. Both CGS and FGS encoding have been proposed for supporting error resilience and simulcasting a multimedia content- through different constraint bandwidths, to different constraint devices, and with different qualities/resolutions. However, we propose to use scalable video encoding in order to support DVD like controls by the user. We use temporal scalability and data partitioning in order to produce playback sequence for interactive playback modes, such as FF and FB. MPEG –7 [92] divides a video into many video sequences, a video sequence into many scenes, and a scene into many shots for video indexing. The metadata about these video segments is generated and used to facilitate video retrieval or browsing. Like MPEG-7, we propose to divide the video into a number of logical segments. We divide a video into several segments in order to provide segment-by-segment DVD like controls.

4.4.1 Temporal Scalability

Playing only every x^{th} frame of a stream we can implement fast forward/backward DVD operations at x speed. In temporal scalable encoding, an original video stream is temporally scaled into a base sub-stream and an enhancement sub-stream, i.e., the video frames at selective temporal positions are placed in the base sub-stream and the remaining frames are placed in the enhancement sub-stream [66]. Therefore, the temporal scalable encoding can also be used to create a sub-stream choosing every x^{th} frame of a stream for FF/FB DVD operations.

In our scheme, a non-scalable encoded stream is first temporally scaled into two sub-streams: base sub-stream and enhancement sub-stream. Base sub-stream consists of all the I-frames and the enhancement sub-stream consists of all the P- and B-frames. So the base sub-stream is kept independent of the enhancement sub-stream and it can be played alone under FF and FB operations. Base sub-stream alone will not produce a good quality video since it is temporally scaled and temporal scalability skips many frames between two consecutive I-frames. Poor video quality is quite acceptable under FF and FB operations. For this reason, we propose to use the above base sub-stream under FF and FB operations.

MPEG-2 encoder uses a parameter X to indicate the interval of two consecutive I-frames, i.e., the size of a GOP. It uses another parameter Y to indicate the interval of two consecutive P-frames as well as the interval of consecutive I-frame and P-frame.The typical values used for Y are 1 to 3. The typical values used for X are 12 to 15 [66]. Using X equals to 12 and Y equals to 3 an MPEG-2 playback stream will look like Figure 4.1.

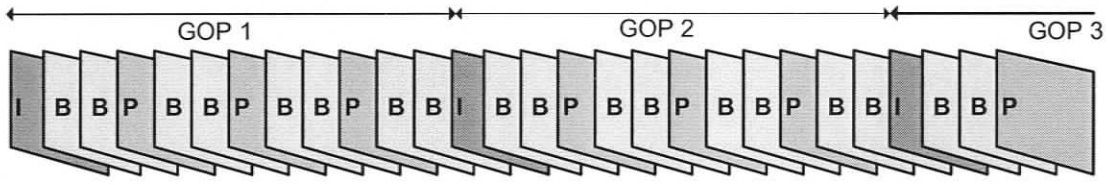

Figure 4.1: Playback stream

After temporally scaling the above stream into two sub-streams, we can get a base sub-stream and an enhancement sub-stream as shown in Figure 4.2.

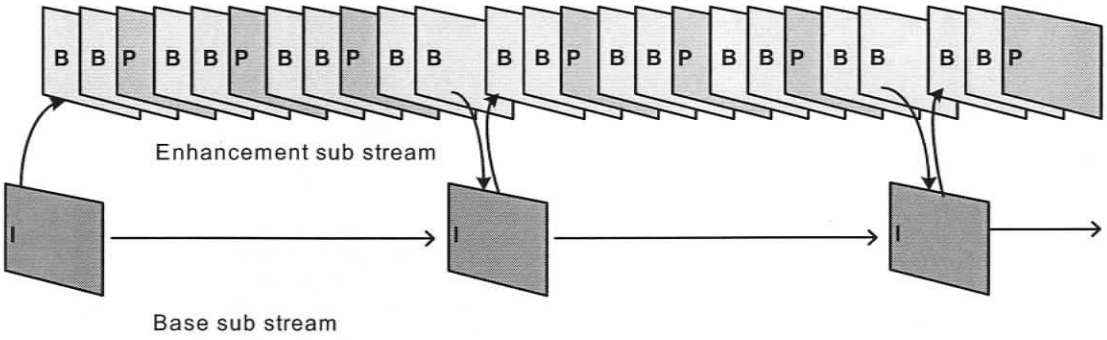

Figure 4.2: Base and enhancement sub-streams after temporal scaling

In normal playback mode, we propose the proxy to combine the base and the enhancement sub-streams if the server performs the above scalable encoding and to transmit the combined stream to the clients. Client decoder can simply decode the frames being transparent to scalable encoding. The use of the above base sub-stream as the FF playback sequence will speed up the playback by a factor of X. We propose the proxy to transmit the base sub-stream in the forward direction in FF playback mode. There are only I-frames in the base sub-stream, i.e., forward prediction is completely absent. This makes the base sub-stream suitable for FB operation too. In FB mode, we propose the proxy to transmit the same base sub-stream in the backward direction. In fact, there is no prediction dependency among the I-frames in the base sub-stream. Selection of any two I-frames in order to make a playback sequence in any direction skipping several I-frames is possible. For this reason, we can also support different FF/FB speeds varying the number of I frames to be skipped from the base sub-stream.

The size of the I-frame is always high compared to other types of frames. It will require a high bandwidth to transmit I-frames during FF/FB operations. High bandwidth requirement to support FF/FB operations is undesirable. To solve this problem, we propose to use data partitioning scalability with temporal scalability.

4.4.2 Hybrid Temporal-Data-partition Scalability

In order to reduce the high bandwidth requirement during the FF/FB DVD operations, we propose to divide the base sub-stream again into two sub-streams using data-partition scalability. In the data-partition scalable encoding, quantized DCT coefficients of a macro block of the original stream is placed in different sub-streams based on the importance of individual coefficient to reconstruct the picture. The DC and the low frequency DCT coefficients of a macro-block are more important than the high frequency coefficients of the same. We propose to place the DC and few low frequency quantized coefficients from the base sub-stream in one sub-stream, which we call low frequency base sub-stream, and the remaining high frequency quantized coefficients in another sub-stream, which we call high frequency base sub-stream. Figure 4.3 shows a macro-block of an I-frame in the base sub-stream before data partitioning. Figure 4.4 shows the corresponding macro-blocks in the low frequency base sub-stream and in the high frequency base sub-stream after data partitioning.

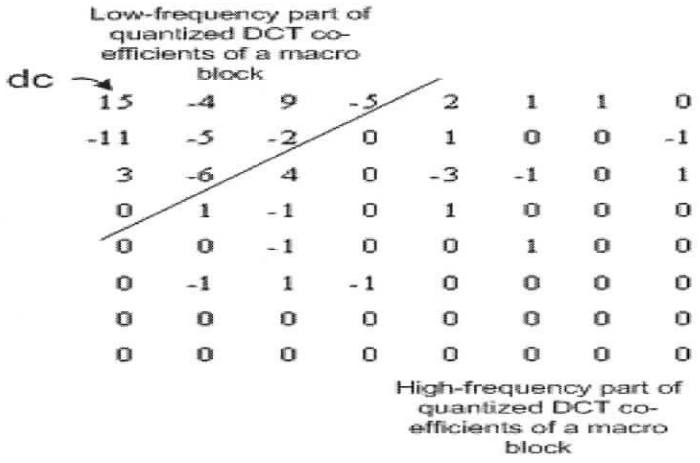

Figure 4.3: A macro block of an I-frame in the base sub-stream before data partitioning

15	-4	9	-5
-11	-5	-2	
3	-6		
0			

Macro-block in the low-frequency base sub-stream

				2	1	1	0
			0	1	0	0	-1
		4	0	-3	-1	0	1
	1	-1	0	1	0	0	0
0	0	-1	0	0	1	0	0
0	-1	1	-1	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Macro-block in the high-frequency base sub-stream

Figure 4.4: Macro blocks of an I-frame in the low frequency base sub-stream and in the high frequency base sub-stream after data partitioning

Adding the high frequency sub-stream with the enhancement sub-stream and leaving the low frequency sub-stream alone as the base sub-stream we can get the resultant enhancement sub-stream and the base sub-stream as shown in Figure 4.5. Encoder's compression efficiency remains high despite the fact that the original stream is divided into two sub-streams.

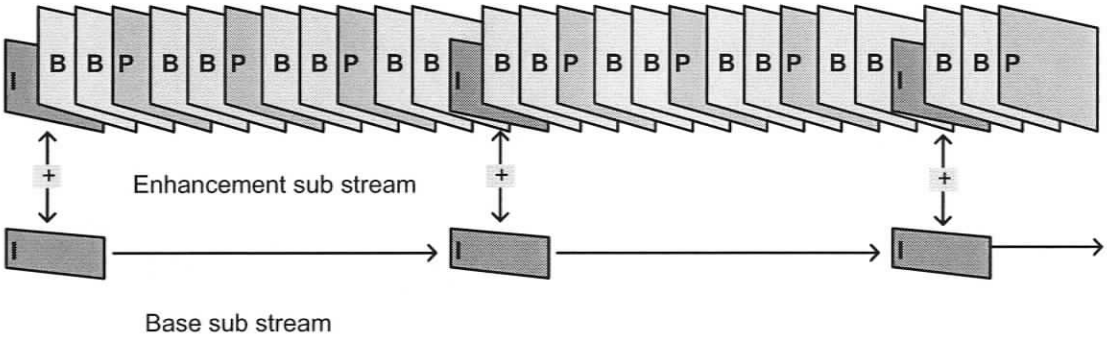

Figure 4.5: Base and enhancement sub-streams after hybrid temporal-data-partition.

The base sub-stream is cached at the proxy. In order to find the memory requirement at the proxy for the base sub-stream, we can compute the average size of the base sub-stream. Typical size of an MPEG-2 I-frame, P-frame and B-frame are 17 kB, 6 kB, and 2.5 kB, respectively [66]. We assume fixed size (number of frames) GOP to make our

computation simple. Temporal scaling reduces the size of the base sub-stream by a factor $\left(1+\frac{1}{3}\left(\frac{X}{Y}-1\right)+\frac{5}{36}\left(X-\frac{X}{Y}\right)\right)$, since it keeps only 1 I-frame but skips $\left(\frac{X}{Y}-1\right)$ P-frames and $\left(X-\frac{X}{Y}\right)$ B-frames from each GOP. It has been mentioned [66] that the DC and few low frequency DCT coefficients of a macro block are enough to get a low resolution but still an acceptable video quality. We can separate the DC and a few, say w, low frequency DCT coefficients in one partition and the remaining high frequency DCT coefficients in another partition. To find an appropriate value for w is still open to research. We have proposed to use the low frequency partition as the base sub-stream, which reduces the size of the base sub-stream by a factor $\left(\frac{64}{w+1}\right)$ in the best case. In the best case, we assume all the DCT coefficients (64) in a macro block are non-zero and only $(w+1)$ of them are taken in the base sub-stream. In the worst case, data partitioning may not reduce the size of the base sub-stream at all. This can happen if all the high frequency coefficients have the value zero in all the I-frames in the base sub-stream. In that case, VLC encoding will compress all those zeros into a single code taking them in a single run and leave nothing to partition. The average case performance is hard to predict. It needs further research to estimate. We assume the best-case performance to make our computation simple. Assuming the best-case performance, the resultant reduction factor is $\frac{64}{(w+1)}\left(\frac{2}{3}+\frac{5}{36}X+\frac{7}{36}\frac{X}{Y}\right)$, i.e., the size of the base sub-stream becomes $\frac{64}{(w+1)}\left(\frac{2}{3}+\frac{5}{36}X+\frac{7}{36}\frac{X}{Y}\right)$ times smaller than that of the original video. For X=12, Y=3 and w=11, the reduction factor is 16.6. Typical size of an MPEG-2 video is about 5GB. Therefore, the base sub-stream will need around 300MB memory at the proxy. Advance memory technology makes it possible to assign this amount of memory at the proxy to cache the base sub-stream of a video. Supported speed up factors are 12, 24, 36, …, x12 for X=12, i.e., X, 2X, 3X, …, xX etc. in general. If the lowest speed up factor is set equal to xX instead of X, the proxy needs only every x^{th} frame instead of every frame from the

base sub-stream to support that speed. Therefore, the size of the base sub-stream will be further reduced by a factor of x. For example, if we set the minimum speed up factor equal to 4x12, the size of the base sub-stream will become 300MB/4 or 75MB. Therefore, we can reduce the memory requirement at the proxy by using the higher speed up factors.

4.4.3 Segmentation and Hybrid Temporal-Data-partition Scalability

MPEG-7 [92] divides a video into many video sequences, a video sequence into many scenes, and a scene into many shots for video indexing. The metadata about these video segments are generated and used to facilitate video retrieval or browsing. It uses *Segment Descriptor Schema* (*Segment DS*) to describe the video sequences, the scenes, and the shots and *SegmentRelation DS* to describe the relationship among video sequences, scenes, and shots [92]. Like MPEG-7, we propose to divide a video into a sequence of several logical segments. The unit of our logical segments can be either a video sequence, or a scene, or a shot. There are many mechanisms described in the literature [94][95][96] to detect shot, scene, and video sequence in a video. These mechanisms are out of the scope of our work and we are not discussing them here. After the segmentation and hybrid scalable encoding is done, a video is comprised of several logical segments and each logical segment S_l is comprised of two sub-streams: enhancement sub-stream $S_{l,e}$ and the base sub-stream $S_{l,b}$, which are shown in Figure 4.6.

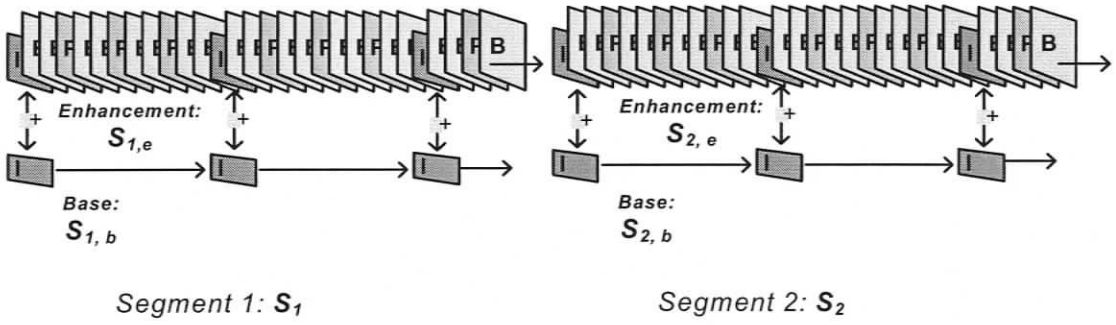

Figure 4.6: Playback sequence of segments and their sub-streams

4.4.4 Location of Segmentation and Scalable Encoding

Either the server or the proxy can perform the above segmentation and hybrid scalable encoding. The amount of data in the prefix as well as in the suffix for each segment depends on the location where the segmentation and the scalable encoding are done. If these are done at the proxy, the server transmits the suffix in the form of original stream (undivided) and the proxy computes the size of the prefix considering the original stream. Thus the amounts of data both in the prefix and in the suffix become high. The proxy caches the prefix and the base sub-stream and buffers the suffix data. Proxy transmits either the cached prefix data or the buffered suffix data (original stream) for normal playback and the base sub-stream data for FF/FB playback. On the other hand, if above segmentation and scalable encoding are done at the server, the server transmits the suffix in the form of enhancement sub-stream leaving the base sub-stream apart. The proxy computes the size of the prefix considering the enhancement sub-stream only. In that case, the amount of data in the prefix as well as in the suffix will be low. Proxy reproduces the original bit-stream combining the enhancement sub-stream (prefix or suffix data) with the pre-fetched base sub-stream and transmits the reproduced bit-stream to the client under normal playback.

4.5. Transmission schemes for different DVD Controls

Different DVD controls need different transmission schemes. In this section, we will describe those transmission schemes briefly.

4.5.1 Transmission scheme for normal Play (Play)

In the normal playback mode, we propose the proxy to start transmitting the video data to the client from the prefix of the first segment and to send a suffix request to the server for the first segment. The proxy transmits the buffered suffix data to the client when the 1st prefix transmission is finished. The proxy sends a suffix request to the server for the next

segment $2d_{s,p}^{\max}$ seconds ahead and starts the prefix transmission of the next segment after finishing the first segment and so on. In this way, continuous transmissions will be going on from the server to the proxy and from the proxy to the client until the whole video is played. If the proxy performs the segmentation and the scalable encoding, the proxy caches the prefix and receives the suffix in the form of undivided stream and the above transmission scheme is sufficient to support normal playback. If the server performs the segmentation and the scalable encoding, the proxy pre-fetches the prefix and receives the suffix in the form enhancement sub-streams. The proxy re-produces the original bit-stream from the base and enhancement sub-streams to transmit it to the client.

4.5.2 Transmission scheme for Forward (JF) and Jump Backward (JB)

In our streaming scheme, the prefix of each segment is always available at the proxy that allows a client to skip its current segment and to resume normal playback from the prefix of any next segment using JF control. The suffix of the jumped segment may already be buffered at the proxy for some other clients. In this case, the proxy allows the jumped client to share the already buffered suffix and resume its normal playback immediately. Otherwise, the proxy sends a suffix request to the server for the jumped segment. Normal playback starts with the prefix and continues with the suffix. A client is allowed to jump back to the beginning of its current segment or to the beginning any previous segment using JB control. When a client resumes its normal playback after a JB operation, the proxy starts transmitting the prefix of the jumped segment and so on.

4.5.3 Transmission scheme for Fast Forward (FF) and Fast Backward (FB)

Our streaming scheme supports various fast forward/backward speeds. There is one minimum speed, which is generally equal to the average number of frames in the GOPs. Other speeds are the multiples of the minimum speed. We propose the proxy to transmit the frames from the pre-fetched base sub-stream in the forward direction under FF operation and in the backward direction under FB operation. If the minimum speed is

chosen, the proxy transmits every frame from the base sub-stream to the client. If a speed x multiple of the minimum speed is chosen, the proxy transmits every x^{th} frame from the base sub-stream. When a FF or FB playback is requested at a point in the video the proxy finds the corresponding frame position in the base sub-stream and starts transmitting the base sub-stream data from that position. After a FF operation, normal playback can be resumed at the beginning of any segment that follows the current segment. Similarly, after a FB operation normal playback can be resumed at the beginning of the current segment or any other segment that precedes the current segment. When the normal playback is resumed, the proxy starts transmitting video data from the pre-fetched prefix of the target segment. The proxy checks, whether the suffix of the target segment is already buffered for some other clients or not. If it is already buffered, the proxy allows sharing the buffered suffix by the interactive client instead of sending the suffix request to the server. Otherwise, it sends a suffix request for the target segment to the server. It transmits the video data from the suffix of the target segment when its prefix transmission is finished.

4.5.4 Transmission scheme for Pause (PA)

Our streaming scheme allows a client to pause its normal playback at any point in the video. The pause duration can be arbitrary. The maximum pause duration can be set a priori too. If the paused point is in a suffix, the proxy maintains the smoothing buffer allocated to the suffix for the paused period. The server continues to transmit the suffix data of the paused segment to the proxy. Proxy buffers the incoming suffix and keeps the suffix data in the smoothing buffer until the paused client has used it after resuming its normal playback. When normal playback is resumed, the proxy starts transmitting the video data from the paused point. The interactive client remains in its own group during and after PA operation.

4.5.5 Transmission scheme for Play Backward (PB)

We have mentioned in Section 2 that to support play backward we need to play reverse predicted video frames in the reverse direction. This is difficult due to the fact that the encoder generally forward predicts the video frames and the proxy receives these frames in the forward direction. In our scheme, video data from the beginning of the current segment up to the current play point remains in the proxy. We propose to use this video data to support PB operation, i.e., PB operation must end at the beginning of the current segment.

We propose to use the straightforward mechanism that has been mentioned in [23] to play forward predicted video frames in the reverse direction. In this mechanism, the proxy will transmit the GOPs from the buffer one after another in the reverse order to the client if a client requests a play backward operation. After receiving a GOP, a client decoder decodes the whole GOP, stores all the decoded frames of the GOP in a buffer, plays decoded frames in the backward direction, and discards the decoded frames from the buffer after they have been played. A client needs a small amount of buffer to hold the decoded frames of a GOP only.

The proxy maintains the smoothing buffer allocated to the current segment and buffers the incoming suffix from the server in the smoothing buffer until the interactive client has used it after resuming the normal playback. When a client resumes normal playback after PB operation, the proxy starts normal transmission of video data from the resuming point. The interactive client remains in its own group at the proxy during and after the play backward operation.

4.5.6 Transmission scheme for Slow Forward (SF) and Slow Backward (SB)

SF operation is almost similar to play normal operation except the difference in their playback rate. SF operation plays the frames at a slower rate than the normal playback rate. Our streaming scheme allows several slow motion rates for SF and SB operations. We propose the proxy to transmit the frames at the selected slow motion rate in the forward direction to support SF operation.

SB operation is also similar to play backward operation. SB operation differs from play backward operation by its playback rate. SB operation plays the frames at a rate slower than that of play backward. We propose to support SB operation almost the same way that we have used to support play backward operation. However, the transmission rate from the proxy to the client for SB operation will be at the selected slow motion rate.

The proxy again, maintains the smoothing buffer allocated to the current segment and buffers the incoming suffix from the server in the smoothing buffer until the interactive client has used it after resuming the normal playback. When a client resumes normal playback after SF or SB operation, the proxy starts normal transmission of video data from the resuming point. Like PA and PB operations, the interactive client remains in its own group at the proxy during and after the SF and SB operations.

4.6. Proxy buffers and server-proxy network bandwidth required to support DVD controls

In our streaming scheme, a proxy sets a base buffer and several prefix buffers, one for each segment, for a video in advance. A proxy allocates smoothing buffers, one for each segment, on demand.

Figure 4.7, shows how the buffers are assigned a priori for the base sub-stream and for all the prefixes of a video, however, only two smoothing buffers are allocated on demand for the video at an arbitrary instance.

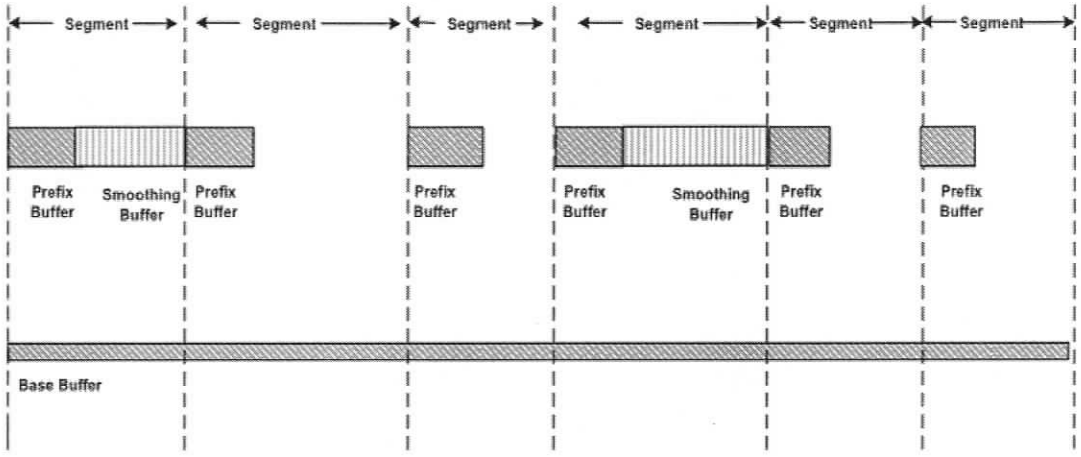

Figure 4.7: Base, prefix, and smoothing buffers for a video at a proxy

The proxy buffers and the server-proxy bandwidth are probably the main constraints on the throughput of a streaming system. For this reason, it is important to know the sizes of the base, prefix and smoothing buffers, and the amount of server bandwidth required for real-time transmission of a video. In this section, we compute the sizes of various proxy buffers and the amount of server-proxy bandwidth required for our proposed streaming scheme.

4.6.1 Base and prefix buffers to support DVD controls

In this section, we compute the sizes of the base and the prefix buffers required at the proxy to support DVD controls, while keeping the streaming service scalable. We assume

the server performs the scalable encoding and the segmentation. Let there be n_s segments in a video and assume that each segment has n_l frames on the average and $f_{j,l,b}$ and $f_{j,l,e}$ be the jth frames of the base sub-stream and the enhancement sub-stream, respectively. Then some frames in the base sub-stream are empty due to temporal scaling, i.e., $f_{j,l,b} = 0$ bits for some j in the base sub-stream. Let the size of the whole base sub-stream be L_b in seconds, i.e.,

$$L_b = \frac{\sum_{l=1}^{n_s} \sum_{j=1}^{n_l} f_{j,l,b}}{b} \qquad (4\text{-}1)$$

Here, b is the mean bit rate of the video in bits per second. We assume that the server transmits the enhancement sub-stream at a fixed rate, which is $r_{s,p}$ times faster/slower than the mean bit rate b of the video. Let the mean frame size of the video be f_{mean} bits, and the frame rate of the video be f_{rate}, where $b = f_{mean} \times f_{rate}$. Let $D_{l,burst}$ denote the burst-work-ahead data of segment S_l. The proxy needs to pre-fetch $D_{l,burst}$ bits of the segment in order to perform work-ahead smoothing (to smooth out the traffic burst of VBR-encoded video). We can find the minimum size of $D_{l,burst}$ from the inequality shown in equation (3-2) in the previous chapter.

If the proxy performs segmentation and scalable encoding, we need to compute $D_{l,burst}$ differently. In that case, we need to consider the frames of the original video stream instead of the frames of the enhancement sub-stream.

The rest of the computation assumes server-side segmentation and scalable encoding. Let $T_{l,prefix}$ seconds denote the size of the prefix of segment S_l. According to equations (3-3) and (3-4) in Chapter 3, the size of the prefix of the first segment will be

$$T_{1,prefix} = \left(2d_{s,p}^{\max} + \frac{D_{1,burst}}{r_{s,p} \times b} \right) \quad (4\text{-}2)$$

Where $d_{s,p}^{\max}$ seconds is the maximum network latency between the server and the proxy. The proxy needs to pre-fetch $2d_{s,p}^{\max}$ additional seconds of data from the first segment to minimize the start up delay. We assume that the proxy always sends consecutive suffix requests to the server $2d_{s,p}^{\max}$ seconds ahead of the playback time and JF, JB, FF, and FB interactions always resume normal playback at least after $2d_{s,p}^{\max}$ seconds, i.e.,

$$T_{l,prefix} = \frac{D_{l,burst}}{r_{s,p} \times b} \qquad \text{for } l > 1 \quad (4\text{-}3)$$

The size of the total prefix buffer at the proxy is thus

$$T_{prefix} = \sum_{l=1}^{n_s} T_{l,prefix} \quad (4\text{-}4)$$

4.6.2 Smoothing buffer to support DVD Controls

From Chapter 3, we know that the size of the smoothing buffer for a segment must be greater than $T_{workahead} = \left(2\Delta_{s,p}^{\max} + \frac{D_{1,burst}}{r_{s,p} \times b} \right)$ seconds in order to overcome jitter and to perform work-ahead smoothing on VBR traffic, where $\Delta_{s,p}^{\max}$ is the maximum jitter, measured in seconds, between the server and the proxy. We assume that each segment suffix is greater than $T_{workahead}$ in order to overcome the jitter and traffic burst problems and to ensure suffix sharing among a group of clients. The proxy allocates a flat smoothing buffer equal to the size of the suffix of the segment when necessary. Therefore, the size of the smoothing buffer for segment S_l is

$$T_{l,smoothing} = \frac{\sum_{j=1}^{n_l} f_{j,l,e}}{r_{s,p} \times b} - T_{l,prefix} \tag{4-5}$$

We compute the average size of the smoothing buffers or the average size of the segment suffixes $T_{average,smoothing}$ in seconds as follows:

$$T_{average,smoothing} = \frac{T_{l,smoothing}}{n_s} \tag{4-6}$$

The number of smoothing buffers needed for a video at a given instant of time depends on the number of segments requested at the proxy at that instant. This number depends on many factors, such as the video's popularity, the probabilities of different DVD interactions on the video, etc. In order to compute the average number of active segments we assume that the request arrival for a given video is a Poisson process with a mean of λ requests/minute. Let the average size of the segments of a video be $E(S)$ seconds and the probability of normal and interactive playback be P_{play} and $(1-P_{play})$ respectively. The probability that there is at least one request for normal playback during $E(S)$ seconds is $\left(1-e^{-\lambda P_{play} E(S)}\right)$. Then the normalized probability that there are s segments requested for a video at the proxy due to normal playback requests is

$$P_n(s) = \frac{\left(1-e^{-\lambda P_{play} E(S)}\right)^s}{\sum_{s=1}^{n_s}\left(1-e^{-\lambda P_{play} E(S)}\right)^s} \tag{4-7}$$

Therefore, the average number of smoothing buffer required at the proxy for normal playback is

$$N_{play,smooth} = \left(\sum_{s=1}^{n_s}\left(P_n(s) \times s\right)\right) \tag{4-8}$$

We assume an interactive playback is initiated on the average after $E(S)/2$ seconds of normal playback. In FF and JF interactions, a client leaves the current segment and resumes normal playback with a different segment. In FB and JB interactive operations, a client may also leave the current segment. If the target segment is not already running, a new smoothing buffer is required to resume normal playback. In play backward, slow forward, and slow backward interactions; a client resumes the normal playback with the current segment. However, it holds the current segment for a longer time than its segment length. On average, this additional holding time is equal to $E(S)/2$ seconds. The Additional holding time for pause interaction is T_{pa} seconds, which is set a priori. We assume T_{pa} is set equal to $E(S)/2$ seconds, to simplify our computation. The probability that there was no normal playback during an interval of $E(S)/2$ seconds for a segment that is needed after any of the above interactive playback modes is $e^{-\lambda P_{play} E(S)/2}$. The probability that there is at least an interactive playback request during an interval of $E(S)$ seconds is $\left(1-e^{-(1-P_{play})\lambda E(S)}\right)$. Then the probability that an additional video segment is required at the proxy for an interactive playback request is

$$P_{\text{int},1} = \left(1 - e^{-(1-P_{play})\lambda E(S)}\right)\left(e^{-\lambda P_{play} E(S)/2}\right) \tag{4-9}$$

The normalized probability that s additional video segments are required due to interactive playback request at the proxy is

$$P_{\text{int}}(s) = \frac{\left(P_{\text{int},1}\right)^s}{\sum_{s=1}^{n_s}\left(P_{\text{int},1}\right)^s} \tag{4-10}$$

Therefore, the additional smoothing buffer requirement for interactive playback at the proxy is

$$N_{\text{int},smooth} = \left(\sum_{s=1}^{n_s}\left(P_{\text{int}}(s)\times s\right)\right) \tag{4-11}$$

Therefore, the total smoothing buffer size $T_{smoothing}$ is

$$T_{smoothing} = (N_{play,smooth} + N_{\text{int},smooth}) \times T_{average,smoothing} \quad (4\text{-}12)$$

Though the sizes of the base buffer and the total size of the prefix buffers are fixed for a particular video at a particular proxy, the total size of the smoothing buffers for a video at proxy varies from time to time based on the demand. In the worst case, all the segments of a video might be active and the proxy will need $n_s \times T_{average,smoothing}$ second smoothing buffer.

4.6.3 Server-proxy network bandwidth to support DVD controls

For each requested segment the server needs to allocate $r_{s,p} \times b$ bps of network bandwidth. The number of segments requested is equal to the number of smoothing buffers allocated due to normal and interactive playback. Therefore, the average server-proxy bandwidth requirement χ bps for a video is

$$\chi = \left(N_{play,smooth} + N_{\text{int},smooth}\right) \times r_{s,p} \times b \quad (4\text{-}13)$$

Therefore, the worst-case server-proxy bandwidth requirement is $(n_s \times r_{s,p} \times b)$ bps. The important point here is to observe that the worst-case buffer and bandwidth requirement does not increase even if the number of clients for a given video is greatly increased. This behaviour of server bandwidth and the proxy buffer requirements ensures that our streaming scheme is highly scalable.

4.7. Simulation results

In order to evaluate our streaming scheme, we have written a simulation program in Java. We have used the video trace of a high quality MPEG-4 encoded movie (Jurassic Park I)

from [80]. It is 60 minutes long; it has 90000 frames; and its frame rate is 25 frames/second. Its mean bit rate is 0.77 Mbps. To make our computation simple we assume that the data partitioning achieves the best-case performance and keeps the DC and 11 low frequency DCT coefficients of the I-frames in the base sub-stream. The size of the base sub-stream is 10.096MB or 1.7568 minutes. We have assumed the maximum propagation delay of 500 ms and the maximum jitter of 250 ms between the server and the proxy. We have divided the video into several logical segments. We have assumed the lengths of these segments are equal to an average segment length in order to make our computation simple. We have used the following default values during our simulation run:

Average request rate $\lambda = 1$ request/minute

Average segment length = 5 minutes

Server-proxy transmission rate ratio $r_{s,p} = 1.0$

Probability of interactive play $(1 - P_{play}) = 0.5$

We plot the total prefix buffer sizes against the average segment length in Figure 4.8. In Figure 4.8, the sum of the prefix buffer sizes fluctuates when the average segment length increases. This is because the video data is not uniformly distributed over the frames and the segments. Figure 4.8 also shows that the sum of the prefix buffer sizes is generally large when the average segment length is small. This is because at a small average segment length, the number of segments, i.e., the number of prefixes becomes large. On the other hand, with a large average segment length the size of the suffix of each segment becomes large, and the proxy needs to allocate large smoothing buffers for these large suffixes of a video. For this reason, the resultant sum of the smoothing buffer sizes increases as the average segment length increases, which are shown in Figure 4.9. The granularity of user control also remains coarse at a large average segment length. We

recommend the use of small average segment length, in recognition of this trade off between small and large segment sizes.

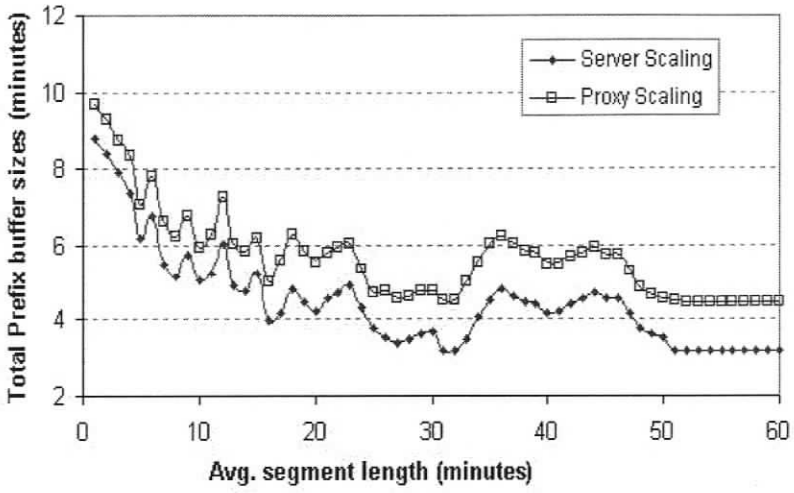

Figure 4.8: Prefix Buffer vs. Segment Length

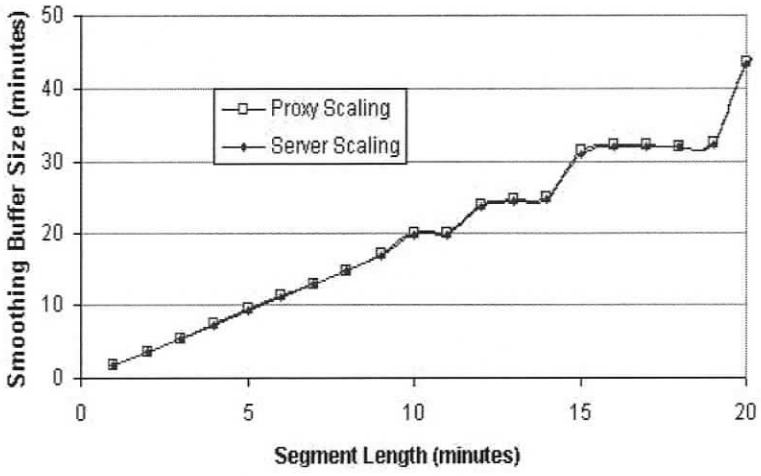

Figure 4.9: Smoothing Buffer vs. Segment Length

Very few segments are required at the proxy when the average request rate is very low. For this reason, the smoothing buffer and the server-proxy bandwidth requirements are very low. If the average request rate continues to increase, it will reach one point where almost all the segments of a video are required at the proxy. For this reason, further increases in the request rate do not increase the buffer and the bandwidth requirements. Figure 4.10 and Figure 4.11 depict the above behaviours, respectively. These behaviours of the smoothing buffer and the bandwidth requirements with respect to request rate show that our streaming scheme is highly scalable. Figure 4.11 shows that the difference between the server-proxy bandwidth requirements for server-side and proxy-side scalable encodings is negligible (both graphs coincide). This is because transmitting (in proxy-side scalable encoding) or not transmitting (in server-side scalable encoding) a small base sub-stream does not make a significant difference in overall bandwidth requirements. We will also see it in Figure 4.13.

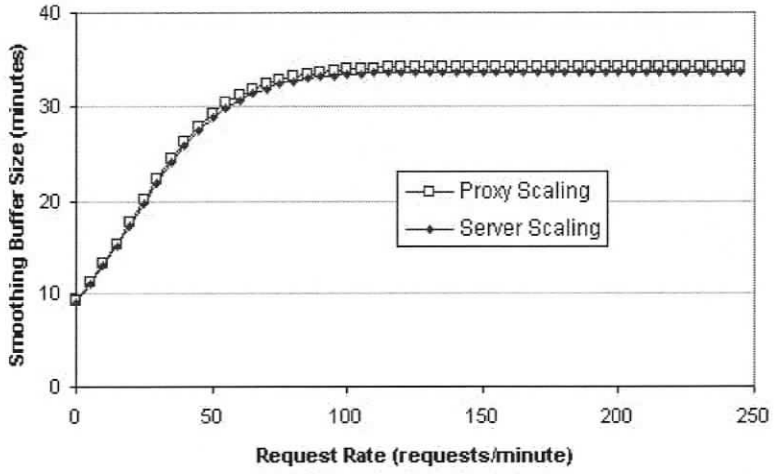

Figure 4.10: Smoothing Buffer vs. Request Rate

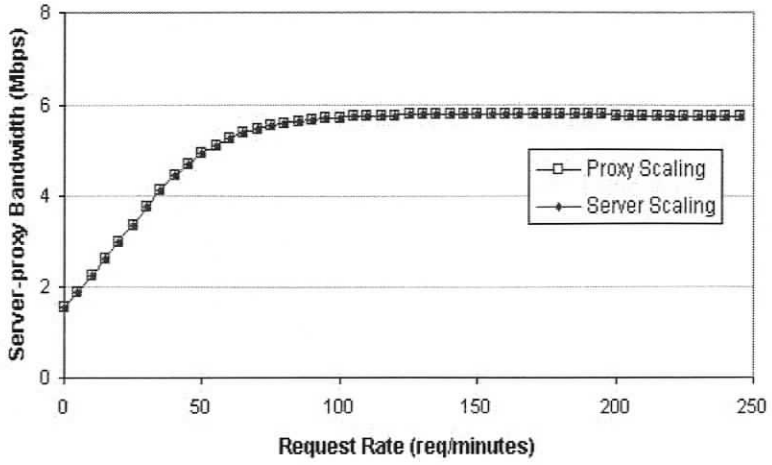

Figure 4.11: Server-proxy Bandwidth vs. Request Rate

We plot the smoothing buffer requirement at the proxy and the bandwidth requirement between the server and the proxy in Figure 4.12 and Figure 4.13 respectively. When the probability of interactive playback is low, most of the playback requests are for normal playback, which needs consecutive segments to be played one after another. This increases the number of segment transmissions from the server and buffered at the proxy. For this reason, at a low interactive playback probability both the server-proxy bandwidth and the smoothing buffer requirements are high. The proxy supports interactive playbacks from the prefix buffer, which is a fixed size buffer with respect to playback probability.

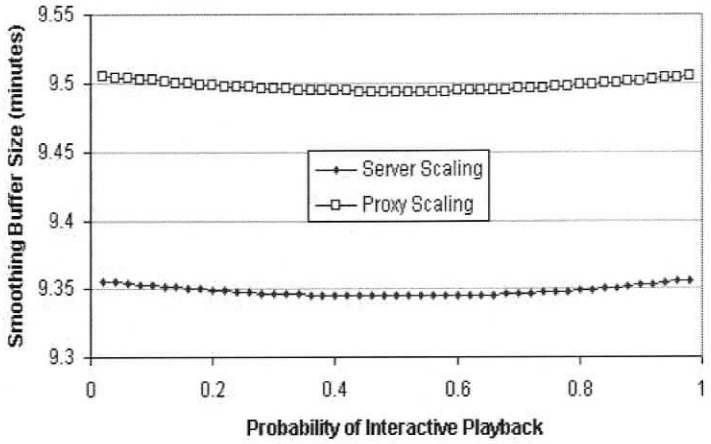

Figure 4.12: Smoothing Buffer vs. Interactive Play

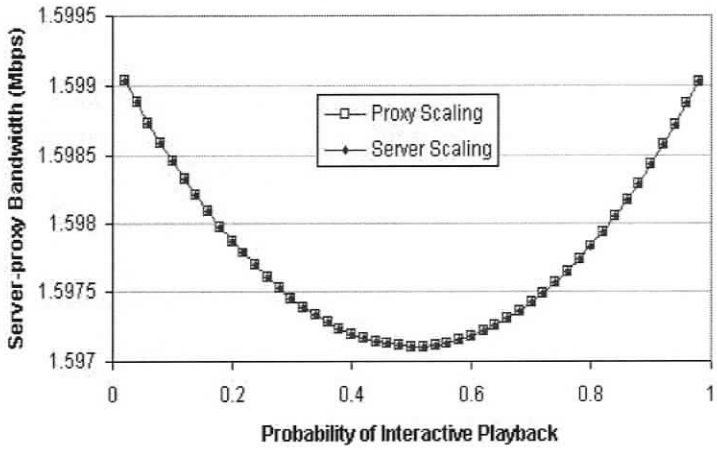

Figure 4.13: Server-proxy Bandwidth vs. Interactive Play

A client may also skip some segments or part of the segments under interactive playback. Both of these facts decrease the expected number of segments required at the proxy, thus reducing the smoothing buffer size, and the server-proxy bandwidth. For this reason, the smoothing buffer requirement at the proxy and the bandwidth requirement between the server and the proxy start to decrease as the probability of interactive playback increases. However, the above requirements stop decreasing after interactive playback probability

becomes 0.5. They, rather, increase after this point. All the interactive playbacks resume normal playback after a while. Normal playback resumptions also need segment suffixes at the proxy. For an interactive playback probability less than 0.5, the chance of the required segment suffixes already being at the proxy buffers for regular normal playbacks is high. For this reason, the number of segment suffixes required for normal playback resumptions after interactive playbacks increases as the interactive playback probability increases after 0.5. The same is true for bandwidth and smoothing buffer requirements. Figure 4.12 and Figure 4.13 also show that the variations in the smoothing buffer and the bandwidth requirements are not much different with respect to interactive playback probabilities. This is a unique feature of our streaming scheme since in other scalable streaming scheme [29] resource requirements increase as the frequency of interactive playback increases.

4.8. Chapter Summary

We have proposed a scalable video streaming scheme that can support segment-by-segment DVD-like user controls for interactive playback modes. Our simulation results show that our streaming scheme remains fully scalable, even when all the clients are highly interactive. It uses a part of the available server-proxy network bandwidth and proxy buffer space to support a large number of clients. It does not increase server-proxy bandwidth and proxy smoothing buffer requirements as the proportion of interactive playback increases. This is because the proxy using the base sub-stream provides interactive playback modes.

Our streaming scheme proposed in this chapter, divides a video into a large number of logical segments and all the segment prefixes are cached at the proxy in order to provide segment-by-segment DVD like interactive playback controls to the users. If the level of interactive playback is low then caching all the segments prefixes at a proxy is highly wasteful. The segmentation of a video is also proposed using an existing algorithm, which

is often complex. In the next chapter, we propose a simple on-demand video segmentation and proxy buffer provisioning scheme in order to avoid buffer over provisioning as well as to avoid the use of complex segmentation algorithm.

5. On-demand Dynamic Video Segmentation and Proxy Buffer Provisioning Scheme

5.1. Introduction

A streaming system needs to provide interactive playback modes for user convenience. A scalable streaming system is required to stream audio/video content to a large number of clients. There are two types of scalable streaming schemes; multicast channel based [62][63] and unicast channel based [98]. Multicast channels are not widely available on the Internet, as the present Internet is mainly unicast enabled. For this reason, unicast based scalable streaming can easily be deployed over the Internet. Unicast based scalable streaming uses one or more proxies between a streaming server and its clients. Both types of scalable streaming schemes share a single server stream among a group of clients. Interactive clients generally ask for different playback sequences that cannot be served by a single shared stream. For this reason, providing interactive playback modes while keeping the streaming scheme scalable is very difficult.

Wang et al. [18] proposed a proxy based scalable streaming scheme for *constant-bit-rate* (CBR) stored videos over the Internet. Their streaming scheme caches the *prefix* or the first few frames of a video in a *prefix buffer* at the proxy. After intercepting a client request, the proxy immediately starts streaming from the prefix buffer and requests the *suffix* or the remaining part of the video from the server. They have assumed that the proxy receives the suffix data just in time when it finishes prefix streaming. If more requests come for the same video before it finishes the prefix streaming, the proxy enables a single suffix sharing among these clients, which makes their streaming scheme scalable. However, the later clients need to receive both prefix and suffix data simultaneously for a while, i.e., they need two channels to receive both prefix and suffix data simultaneously. Just in time arrival of suffix data cannot be guaranteed in the

Internet. *Variable-bit-rate* (VBR) encoded videos cannot be streamed in their scheme. Interactive playback is not possible in their streaming scheme either.

In Chapter 3, we proposed a proxy based scalable streaming scheme for both CBR- and VBR-encoded videos over the Internet. Along with the prefix buffer we used a *smoothing buffer* at the proxy to buffer the suffix data temporarily. In our scheme, the proxy would start streaming from the smoothing buffer after finishing streaming from the prefix buffer, i.e., a single channel between the proxy and a client is sufficient. We have used smoothing buffer to share a single suffix among a group of clients, who are requesting the same video within a predefined sharing interval. We also used the prefix and the smoothing buffers to smooth out network jitter and the traffic bursts of VBR-encoded videos. We enabled CBR-transmission of VBR-encoded videos both from the server to the proxy and from the proxy to the client. However, interactive playback is not possible in the streaming scheme as proposed in Chapter 3.

In Chapter 4, we proposed another proxy based scalable and interactive streaming scheme over the Internet. We have proposed to divide a video into several logical segments; each segment into a prefix and a suffix; and to cache all the prefixes at the proxy. We have proposed to create an interactive playback sequence for interactive playback modes using scalable encoding. The streaming scheme proposed in Chapter 4 provides DVD like interactive playback controls to the clients. It also provides interactive playback modes while keeping the scheme scalable. However, segmentation of a video is supposed to be done using a complex segmentation algorithm. Such complex algorithms have been proposed in [95][96]. All the segment-prefixes of a video are cached at the proxy a priori assuming that there will be a high level of interactive playback demand for the video. If the level of the demand for interactive playback is not high then the above provisioning is a straight wastage of the resources.

In this chapter, we propose a simple on demand video segmentation and proxy buffer provisioning scheme for an interactive and scalable video streaming system in order to

avoid buffer over provisioning as well as to avoid the use of complex segmentation algorithms. In this scheme, the proxy starts with a single prefix and a single suffix of a video, i.e., with a non-segmented video and with a minimally provisioned buffer similar to a streaming scheme of Chapter 3. Segmentation, segment-prefix computation, and segment-prefix caching are done on demand. The proxy gradually generates more segments and caches more segment-prefixes as it receives more interactive playback requests. And this segment generation process stops at some point when an adequate number of segments have already been generated to meet a high level of interactive playback demand. We call this point *segmentation saturation point*. It avoids over provisioning of proxy-buffer for a video by gradually dividing the video into many segments and allocating prefix buffers for these new segments as the number of interactive requests increases. Segments are generated using client's playback behaviours; does not need to run complex scene/shot/frame detection algorithms for segmentation; and does not need to divide segments blindly on media time. These points make our scheme considerably attractive and beneficial.

The rest of the chapter is organized as follows. In Section 5.2, we describe our on demand segmentation and buffer-provisioning scheme. In Section 5.3, we discuss what minimum segment length we should use, which is important for server-proxy bandwidth and proxy-buffer management. In Section 5.4 we compute the average number of segments being created during a full playback of a video and the time required to reach the segmentation saturation point of a video. We compute the total prefix and smoothing buffer requirement at the proxy for a video in Section 5.5. We compute the total server-proxy bandwidth requirement for a video in Section 5.6. Simulation results are shown in Section 5.7. We conclude the chapter in Section 5.8.

5.2. Our Proposed On-demand Segmentation and Buffer Provisioning

We propose a proxy based scalable and interactive streaming scheme to start streaming with a non-segmented video. At first, a video is divided into only one prefix and only one suffix. The mechanism to divide a video into appropriate prefix and suffix has been described in Chapter 3. A *base sub-stream* of the video is created using hybrid temporal-data-partitioning scalable encoding as described in Chapter 4 as the playback sequence for fast-forward/backward playback mode. The proxy caches the prefix and the base sub-stream. When a client requests normal playback of a video, the proxy starts transmitting the video data from the prefix buffer and sends a suffix request to the server. The proxy buffers incoming suffix data in a smoothing buffer and starts streaming from the smoothing buffer after finishing streaming from the prefix buffer. A group of clients, who are requesting the same video within a *sharing interval* of T_{share} seconds, shares a single suffix and a smoothing buffer. Prefix and smoothing buffers are also used to smooth out network jitter and the traffic bursts of VBR-encoded videos. They also enable CBR-transmission of VBR-encoded videos from the server. The smoothing buffer remains allocated to a group of clients until the playback of the video ends or the group is empty. The group may become empty if all the clients leave the group due to interactive playback. The proxy may allocate several smoothing buffers for several groups of clients. Figure 5.1 depicts such a scenario. In Figure 5.1, two smoothing buffers have been allocated on demand for two groups of clients. A single prefix buffer and a single base sub-stream are used for all the clients. The *base buffer* is used to cache the base sub-stream for interactive playback modes.

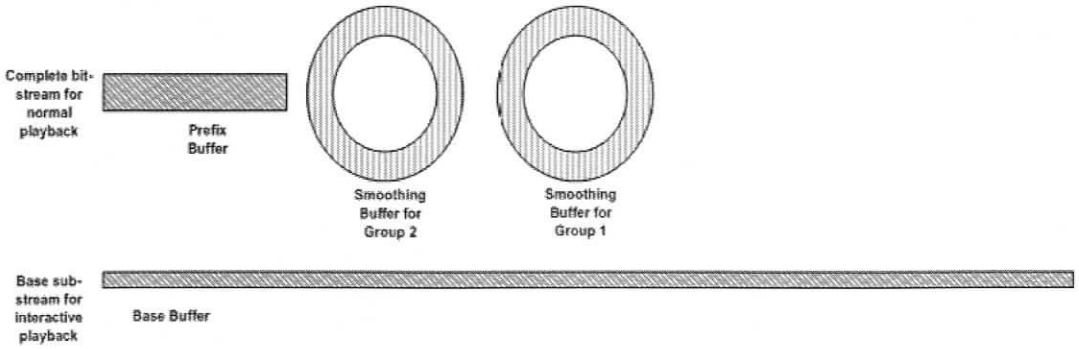

Figure 5.1: Streaming: non-segmented and minimally provisioned

When a client requests *fast-forward* (FF) or *fast-backward* (FB) playback the proxy splits off of the client from its group and starts transmitting base sub-stream data from a corresponding point, where the client interrupts normal playback, in the base sub-stream. Proxy transmits the base sub-stream data in the forward direction for FF playback mode and in the reverse direction for FB playback mode. The client uses the base sub-stream data as the playback sequence for FF and FB playback modes. *Jump-forward* (JF) and *jump-backward* (JB) playback modes do not need playback sequence and the proxy does not need to transmit any data during JF and JB playback modes though it splits off the client from its current group. Initially, there is only one segment comprised of the whole video and a client is allowed to resume normal playback at any point he/she wants to. If there is a group of clients already running at the point where the client wants to resume its normal playback, it will re-start normal playback right away by joining in that running group. Otherwise, the client has to wait to re-start normal playback until the proxy finishes segmentation, segment-prefix computation, and segment-prefix caching. Only the first client, who resumes normal playback in a region with potential for segmentation, creates new segments and waits. A region has potential for segmentation, if a new segment can start at any frame in that region. The length of a potential region is equal to *minimum–segment-length*, which is set a priori for a video. The minimum-segment-length is the least possible length of a segment of a video. We describe how to select an appropriate value for the minimum segment length in Section 5.3. Later clients who want

to resume normal playback in the same region, do not create new segments and do not wait. It is allowed to create only one new segment starting in a potential region. When the first client resumes its normal playback in the region, the proxy divides the current segment into two segments at the resuming point. The segment at the left side ends just before the resuming point and the segment at the right side starts from the resuming point. The proxy computes and allocates the required sizes of the prefix and the smoothing buffers for these new segments. It copies the necessary amount of data from the old prefix buffer into the new left prefix buffer and de-allocates the old prefix buffer. If the interactive client joins in a running group, the proxy caches the necessary amount of data from the smoothing buffer of the running group into the new right prefix buffer. Otherwise, the proxy sends a data request to the server and allocates a new smoothing buffer. This new smoothing buffer is also shared among other clients, whose play points are within a threshold distance of T_{share} seconds. In this case, the proxy first caches the incoming data into the right prefix buffer. The interactive client waits to restart normal playback until the above prefix caching is complete in order to ensure smooth playback of the segment. Once the prefix caching is complete the proxy starts transmitting video data from the prefix buffer and buffers the incoming data from the server into the new smoothing buffer.

In Figure 5.2, we see that one segment is divided into two new segments when one client resumes normal playback after a FF playback and no segment begins less than the minimum-segment-length distance away from the intended resuming point. In this case, the actual resuming point and the intended resuming point are the same. A new prefix buffer is allocated to each new segment. A new smoothing buffer is also allocated since there is no running group at the resuming point.

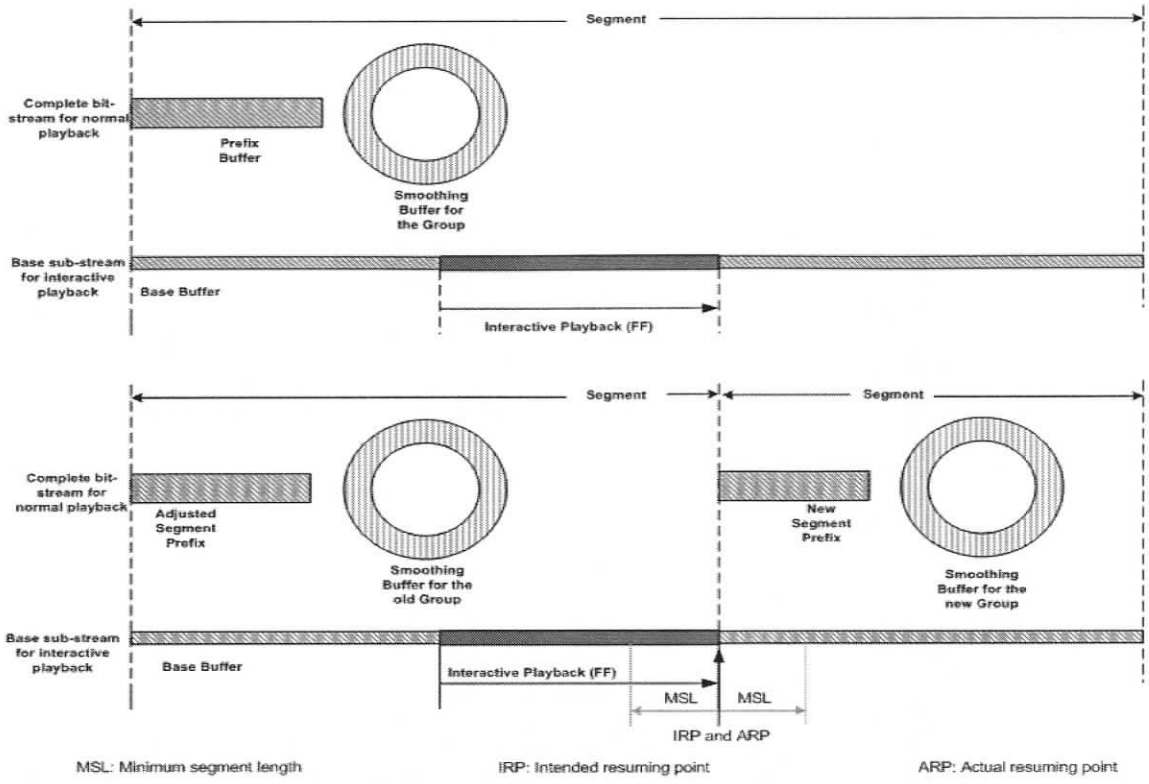

Figure 5.2: One segment is divided into two segments after FF playback.

If there is already the start of an old segment at the left, whose distance is less than the minimum-segment-length from the intended resuming point, the client resumes its normal playback from the beginning of the old segment instead of the intended resuming point. In this case, the proxy does not create a new segment. Figure 5.3 shows this case.

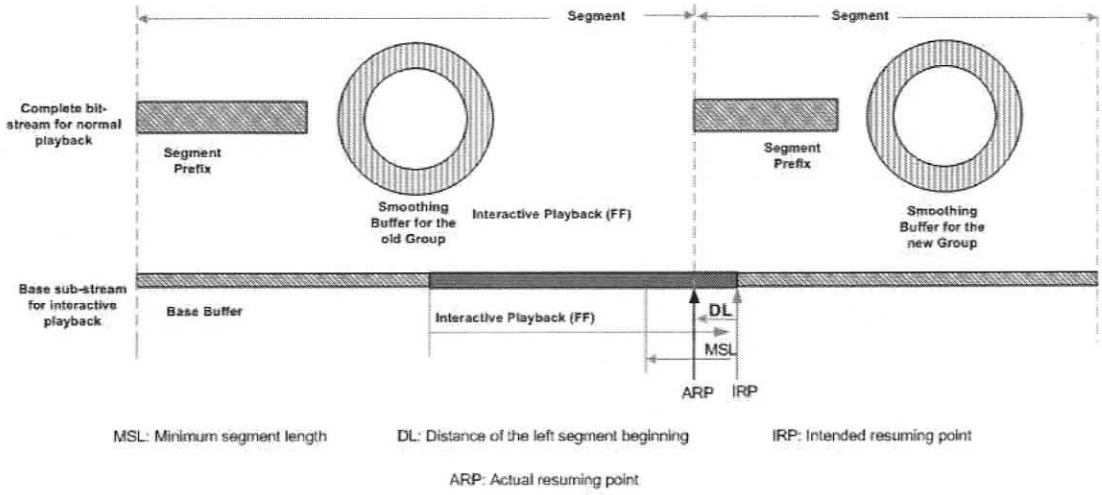

Figure 5.3: An old segment starts at a very small distance away at the left side of the intended resuming point

If the start of an old segment is at the right side of the intended resuming point at a distance less than the minimum-segment-length, a new resuming point is chosen at the left of the old segment at a distance equal to the minimum-segment-length from the beginning of the old segment. If there is no other old segment starting at the left of the new intended resuming point at a distance less than the minimum-segment-length, a new segment is created at the new resuming point. A new prefix buffer is allocated to the new segment. A new smoothing buffer is also allocated if there is no running group at the resuming point. Figure 5.4 depicts such a scenario. However, if there is another old segment starting at the left of the new intended resuming point at a distance less than the minimum-segment-length, the client resumes its normal playback from the beginning of this old segment instead of the new intended resuming point and the proxy does not create a new segment. Figure 5.5 shows such a case.

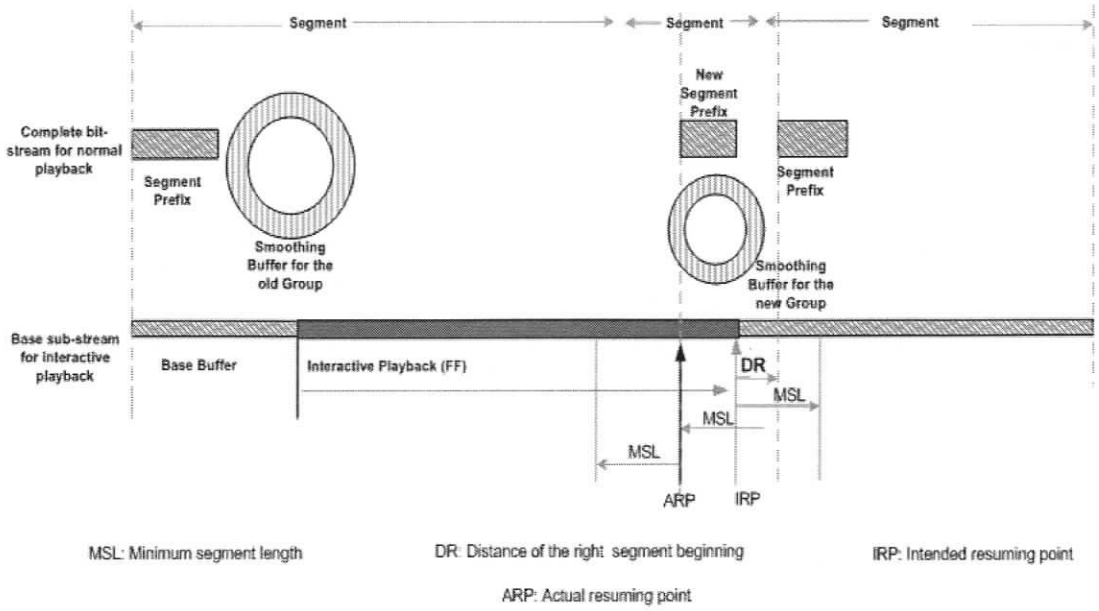

Figure 5.4: An old segment starts at a very small distance away at the right side of the intended resuming point and there is no other nearby old segment at the left of that old segment.

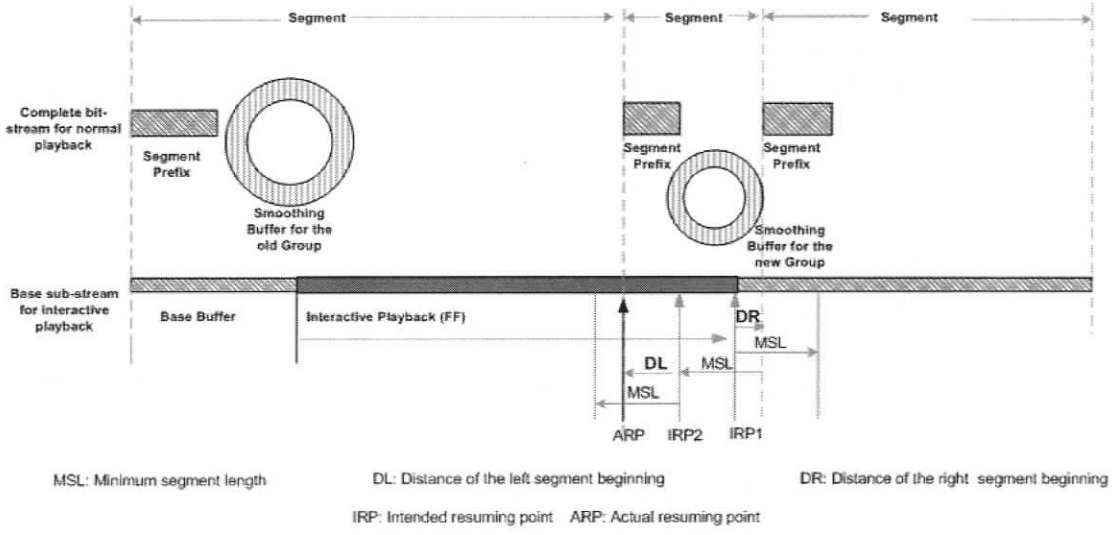

Figure 5.5: An old segment starts at a very small distance away at the right side of the intended resuming point and there is another nearby old segment at the left of that old segment.

The complete pseudo-code for finding the actual resuming point and to create new segments, based on an intended resuming point is given below:

```
/* Global variables */
min-frames                  /*the minimum number of frames that must present in a segment */
list-of-segments    /* the list of segments that have already been created for the video */

/* resume algorithm decides the frame position from where normal playback should re-start based on an intended
resuming point and creates new segments if necessary.
 */
resume (intended-point, current-segment) {
        left-segment=max((segment in the list-of-segments whose starting point)≤intended-point)
        right-segment=min((segment in the list-of-segment whose starting point)≥intended-point)
        Compute the distance of the intended-point from the left-segment
        Compute the distance of the intended-point from the right-segment
        if (distance from left-segment < min-frames) {
                No new segment
                Resume normal playback from the beginning of the left-segment
                }
        else if (distance from right-segment < min-frames) {
                Compute the new intended-point = right-segment -- min-frames
                Find the new left-segment= max ((segment in the list-of-segments whose starting point) ≤ new
                intended-point)
                Compute the distance of the new intended-point from the new left-segment
                if (distance from the new left-segment<min-frames) {
                        No new segment
                        Resume normal playback from the beginning of the new left-segment
                        }
                else {
                        Create a new segment, which starts from the beginning of the current-segment and ends
                        at the new intended-point
                        Create another new segment, which starts from the new intended-point and ends at the
                        end of the current-segment
                        Remove the current-segment
                        }
                }
        else {
                Create a new segment, which starts from the beginning of the current-segment and ends at the
                intended-point
                Create another new segment, which starts from the intended-point and ends at the end of the
                current-segment
                Remove the current-segment
                }
        }
```

If demands for interactive playback modes for a video are high, the video will be divided gradually into many segments and the prefix of those segments will also be cached gradually. A non-segmented and minimally provisioned streaming scheme will be transformed into a highly segmented and a well provisioned streaming scheme. Clearly, the level of segmentation and provisioning depends on the level of interactive playback demand. Figure 5.6 shows a highly segmented and highly provisioned video. In Figure 5.6, most of the segments became minimum-length segment and their smoothing buffers became flat.

The smoothing buffer is a circular buffer at the beginning and it remains circular as long as the length of the corresponding segment is greater than the minimum-segment-length. After successive segmentations the length of an individual segment may become equal to the minimum segment length. Then the smoothing buffer of that segment becomes flat. In a circular smoothing buffer, previously buffered video data is often overwritten by the incoming data when the buffer is full. All the clients who are sharing a smoothing buffer must play the buffered data before it is overwritten.

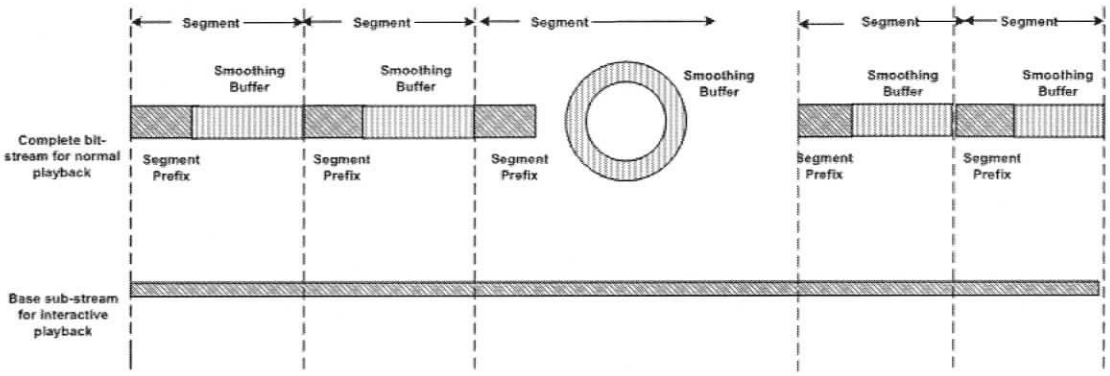

Figure 5.6: A highly segmented and provisioned video with flat smoothing buffers

Therefore, the life of the video data in a circular smoothing buffer cannot be extended by slowing down the playback rate, which is necessary for slow motion playback modes,

such as *slow-forward* (SF) and *slow-backward* (SB). For this reason, the proxy does not provide slow motion playback modes on a video segment until the smoothing buffer of that segment becomes flat. Recently played video frames may also be lost from a circular smoothing buffer due to buffer overwriting. For this reason, the proxy cannot provide *play-backward* (PB) playback mode on a segment as long as the smoothing buffer of that segment remains circular. In a flat smoothing buffer, buffer overwriting does not happen. Recently played video frames stay in the flat smoothing buffer and the life of the buffered data can also be extended. Therefore, the proxy can provide SF, SB, and PB playback modes on a segment with a flat smoothing buffer. For this reason, proposed scheme does not provide SF, SB and PB playback modes initially though it provides Play normal, FF, FB, JF, and JB playback modes always. It starts providing SF, SB, and PB playback modes on a segment as soon as the smoothing buffer of that segment becomes flat.

5.3. The Minimum-Segment-Length

In our proposed streaming scheme, each segment of a video is divided into a prefix and a suffix, which is the remainder of the segment. The prefix is cached at a proxy in a prefix buffer. The suffix is transmitted from the server on demand and buffered in a smoothing buffer at the proxy. It has been shown in Chapter 3 (Section 3.3.4) that the size of the prefix buffer depends on the delay and the delay jitter between the server and the proxy as well as on the level of the bursts in the video data. The size of the smoothing buffer depends on the size of the work-ahead buffer and on the sharing interval T_{share}, during which the requests share the same smoothing buffer and the same server stream. The size of the work-ahead buffer, again, depends on the size of the prefix buffer. If a segment is large its smoothing buffer behaves like a circular buffer and a proxy may need to allocate more than one smoothing buffers for that segment and the server may need to transmit the segment many times within a single segment duration. On the other hand, if a segment is small, i.e., its size is equal to the minimum-segment-length, its smoothing buffer behaves like a flat buffer, and one smoothing buffer at the proxy and one segment transmission from the server within the segment duration is sufficient. From the above

discussion it is clear that the proxy can use the segment length as a tool for its buffer and server-proxy bandwidth management. Using a large value for the minimum-segment-length requires fewer streams from the server and expedites circular to flat buffer transformation, i.e., leads to less smoothing buffer requirement. However, it increases the average smoothing buffer size as well as the average holding time of these large smoothing buffers at the proxy. These will cause poor buffer utilization at the proxy. Moreover, large minimum-segment-length means fewer segments of a video, which will cause a coarse granularity of interactive playback. We propose to choose a not too large and not too small segment length in order to get better buffer utilization and a satisfactory level of interactive playback granularity.

In Chapter 3 (Section 3.3.4), the size of the prefix buffer has been computed in terms of network latency, jitter, and traffic bursts. The proxy pre-fetches data in the prefix buffer in order to smooth out network jitter and traffic bursts through *work-ahead smoothing*. We call the amount of data required to smooth out traffic bursts *burst--work-ahead data* D_{burst}. The size of D_{burst} has been computed from the following expression.

$$\forall j\left[\left(j\in(1,\ldots,n)\right)\Lambda\left(\left(\sum_{k=1}^{j} f_k\right)\le\left(j\times r_{s,p}\times f_{mean}+D_{burst}\right)\right)\right] \qquad (5\text{-}1)$$

Here, f_j is the size of the j^{th} frame in bits, f_{mean} is the mean frame size, n is the total number of frames of a VBR-encoded video, and $r_{s,p}$ is the ratio between the transmission rate and the mean encoding bit rate of the video. We have assumed that the minimum *server-to-proxy network latency* is $d_{s,p}^{\min}$ seconds, and the maximum *server-to-proxy jitter* is $\Delta_{s,p}^{\max}$. In order to overcome server-to-proxy latency and to smooth out network jitter and traffic bursts the size of the prefix buffer in seconds has been computed in Chapter 3 as follows:

$$T_{prefix} = (2d_{s,p}^{\min} + 2\Delta_{s,p}^{\max} + \frac{D_{burst}}{r_{s,p} \times b}). \quad (5\text{-}2)$$

A portion of the smoothing buffer is used to perform work-ahead smoothing in order to overcome the problems due to network jitter and the traffic bursts of VBR-encoded video. We call this portion of smoothing buffer *work-ahead smoothing buffer*. The size of the work-ahead smoothing buffer $T_{workahead}$, and the size of the smoothing buffer $T_{smoothing}$ for a video have been computed in Chapter 3 as follows:

$$T_{workahead} = \left(2\Delta_{s,p}^{\max} + \frac{D_{burst}}{r_{s,p} \times b} \right) \quad (5\text{-}3)$$

$$T_{smoothing} = (T_{share} + T_{workahead}) \quad (5\text{-}4)$$

Here, $T_{workahead}$ and $T_{smoothing}$ have been expressed in seconds.

The prefix and the work-ahead smoothing buffers are used to overcome delay, delay jitter, and data burst problems; they do not contribute in stream and buffer sharing. Only T_{share} portion of a smoothing buffer contributes in stream and buffer sharing among many clients. The larger the value of T_{share}, the more clients can share the same stream and the same buffer at the proxy. To minimize the overheads of the prefix and the work-ahead buffers it is necessary to choose a value for T_{share} larger than the summation of the sizes of the prefix and the work-ahead buffers, i.e.,

$$T_{share} \geq (T_{prefix} + T_{workahead}) \quad (5\text{-}5)$$

In order to ensure above smoothing and sharing it is obvious that the minimum-segment-length must be greater than or equal to the summation of the sizes of the prefix and the smoothing buffers. Let the minimum-segment-length be $S_{\min,length}$ seconds, i.e.,

$$S_{\min,length} \geq (T_{prefix} + T_{smoothing}),$$

$$\text{i.e., } S_{\min,length} \geq \left(T_{prefix} + T_{workahead} + T_{share}\right) \quad (5\text{-}6)$$

We have already mentioned that the smoothing buffer of a segment becomes flat if its length becomes $S_{\min,length}$ seconds. Then the whole segment can be shared among the clients who have requested the segment within an interval of $S_{\min,length}$ seconds, i.e., the effective sharing interval changes from T_{share} seconds to $S_{\min,length}$ seconds. This is why at the minimum-segment-length only one smoothing buffer allocation at the proxy and only one server stream for a segment is sufficient. Expressions (5-2) and (5-3) show that the values of T_{prefix} and $T_{workahead}$ are fixed for a particular video and for a particular proxy. However, a proxy can adjust the value of $S_{\min,length}$ by adjusting the value of T_{share} in order to perform its buffer and server-proxy bandwidth management.

5.4. Average Number of Segments and average Segmentation Time

The number of segments of a video is important for many reasons. First, it determines how many prefix buffers are necessary at the proxy for the video. Secondly, the number of server-proxy transmissions or the amount of server-proxy bandwidth depends on it. Thirdly, it determines the granularity of interactive operations as well as the total number of supported interactive playback modes. For example, most of the segments remain large and their smoothing buffers remain circular if a video has few segments; the proxy cannot support SF, SB, and PB playback modes on these large segments; and an interactive playback can end at any arbitrary position on these large segments. On the other hand, if a video has a large number of segments then most of the segments become equal to the minimum-length segment and their smoothing buffers become flat. The proxy provides SF, SB, and PB playback modes on these small segments. An interactive playback is most likely to end at the beginning of a small segment instead of an arbitrary position in a large segment. A video also needs a large number of prefix buffers for a large number of segments. A large number of segments do not have any merit for normal

playback and are only beneficial for interactive playback modes. For this reason, we have proposed to create more segments of a video if and only if it receives a large number of interactive playback requests. Therefore, the number of segments of a video in our scheme depends on the number of interactive playback requests. In this section we compute the average number of segments as the function of interactive playback request rate.

Like Chapter 3, we assume that the arrival of the playback requests for a video is a Poisson process with a rate equal to λ requests/s. Like Chapter 4, we assume that the probability that a request is for normal playback is P_{normal}, i.e., the probability that a request is for interactive playback is $(1-P_{normal})$. It is obvious that the arrival of interactive playback request is also a Poisson process with a rate $(1-P_{normal})\lambda$ requests/s. The probability that there is no interactive request for the video at an instant is $\left(e^{-(1-P_{normal})\lambda}\right)$. And the probability that there is only one interactive request at an instant is $(1-P_{normal})\lambda e^{-\lambda(1-P_{normal})}$. Let the length of the video expressed in time be L seconds. Then the relation between L and n is $L=\frac{n}{f_{rate}}$, where f_{rate} is the *frame rate* of the video expressed as frames/s. We can say that the resuming normal playback at a frame follows Uniform distribution. Then the probability that interactive clients resume normal playback at a particular frame is $\frac{1}{n}$ and the probability that interactive clients resume normal playback at any one frame of a specific group of m consecutive frames is $\left(\frac{m}{n}\right)$. Similarly, the probability that interactive clients do not resume normal playback at any of the frames of a specific group of m consecutive frames is $\left(1-\frac{m}{n}\right)$. Let the minimum-segment-length be m frames when it is expressed in terms of the number of frames, then $S_{\min,length}=\frac{m}{f_{rate}}$. Then the number of frames in a region potential for segmentation is m.

We can say that the probability that there are two or more resumptions of normal playback at an instance is $\left(1-e^{-(1-P_{normal})\lambda}-(1-P_{normal})\lambda e^{-(1-P_{normal})\lambda}\right)$. Then the probability that a new segment is being created is a joint probability of two probabilities. One of them is the probability of normal playback resumption in a potential segmentation region, i.e., $\left(\frac{m}{n}\right)$. And the other one is the probability of no other resumption of normal playback happened in the same region, i.e., $\left(1-\frac{m}{n}\right)\left(1-e^{-(1-P_{normal})\lambda}-(1-P_{normal})\lambda e^{-(1-P_{normal})\lambda}\right)$. These probabilities are independent of each other. Therefore, the probability that one segment is being created is

$$P_{seg1}=\left(\frac{m}{n}\right)\times\left(1-\frac{m}{n}\right)\times\left(1-e^{-(1-P_{normal})\lambda}-(1-P_{normal})\lambda e^{1(1-P_{normal})\lambda}\right) \qquad (5\text{-}7)$$

Segments are created on demand continuously. It might reach segmentation saturation point where no more segmentation is possible, i.e., the maximum possible number of segments of a video has already been created. When two new segments are created at the middle of an old segment of length $2S_{\min,length}$ seconds their lengths become $S_{\min,length}$ seconds. Since this by-section is a probabilistic event, all the segments at saturation might not be $S_{\min,length}$ second long. Some of them may be greater than $S_{\min,length}$ seconds but less than $2S_{\min,length}$ seconds. However, in the worst case, we can assume that all the segments are $S_{\min,length}$ second long. Therefore, the possible *maximum number of segments* for the video is

$$S_{\max,number}=\left\lceil\frac{L}{S_{\min,length}}\right\rceil \qquad (5\text{-}8)$$

Where the operation $\lceil\ \rceil$ returns the minimum integer greater than or equal to its operand.

According to Poisson probability distribution a video has $I = \lfloor (1 - P_{normal})\lambda L \rfloor$ interactive playback requests in a single video duration, where the operation $\lfloor \ \rfloor$ returns the maximum integer less than or equal to its operand. We have assumed Poisson distribution for the arrival of video requests, since there is a known average request rate for each video and the number of requests arrive in the current interval does not depend on the number of requests arrived in the previous intervals. We assume that the creation of new segments by interactive playback requests follows Binomial probability distribution, since we know the total number of interactive requests for a video during its full playback time and the probability that each interactive request creates a new segment. Then the probability that s segments have been created for the video during L seconds is

$$P_{seg,s} = \binom{I}{s} (P_{seg,1})^s (1 - P_{seg,1})^{I-s} \tag{5-9}$$

Finally, the *average number of segments* $S_{avg,number}$ created for the video in L seconds can be computed as

$$S_{avg,number} = 1 + \begin{cases} \sum_{s=1}^{S_{max,number}-1} (P_{seg,s} \times s) & if \lfloor ((1 - P_{normal})\lambda L) \rfloor \geq (S_{\max,Number} - 1) \\ \sum_{s=1}^{\lfloor (1-P_{normal})\lambda L \rfloor} (P_{seg,s} \times s) & otherwise \end{cases} \tag{5-10}$$

And the *average segmentation time,* S_{time} seconds, which is the time necessary to reach the segmentation saturation point, can be computed as

$$S_{time} = \left(\frac{S_{\max,number} - 1}{S_{avg,number} - 1} \right) \times L \tag{5-11}$$

5.5. Buffer Requirement at the Proxy

We have computed the average number of segments for a video in Section 5.4 and the minimum segment length in Section 5.3. In Section 5.3, we have also mentioned that the

proxy can adjust the value of the minimum segment length $S_{\min,length}$ by adjusting the value of the sharing interval T_{share} in order to get better buffer and server-proxy bandwidth utilization. For this reason, the proxy needs to know the total buffer requirement for a video at a particular choice of sharing interval. In this section, we compute the total buffer requirement for a video as a function of sharing interval.

If the average segment length is represented by $S_{avg,length}$ and is expressed in seconds, we can write $S_{avg,length} = \frac{L}{\min(i \times S_{avg,number}, S_{\max,number})}$ after $(i \times L)$ seconds. This is because in each L second $S_{avg,number}$ segments are created and more than $S_{\max,number}$ segments cannot be created for the video. In order to make our computation simple, we assume that all the smoothing buffers are circular if $S_{avg,length} > S_{\min,length}$ and all the smoothing buffers are flat if $S_{avg,length} \leq S_{\min,length}$. For this reason, buffer computation will be different for these two different conditions. At first, we compute the buffer requirement when $S_{avg,length} > S_{\min,length}$. In this case, the buffer is shared among the clients who have requested the same video within an interval of T_{share} seconds. If there is at least one request for the video in a time slot of T_{share} seconds, a smoothing buffer is allocated for the video. We assume that the normal playback requests are coming at a rate λP_{normal} requests/s. Therefore, the probability that a smoothing buffer has been allocated for normal playback of a video in a time slot T_{share} is

$$P_{n\_smooth,1} = \left(1 - e^{-\lambda P_{normal} \times T_{share}}\right) \tag{5-12}$$

There are $S_{slot} = \frac{S_{avg,length}}{T_{share}}$ such slots in a segment. According to Binomial probability distribution, the probability that i smoothing buffers have been allocated for a segment is

$$P_{n\_smooth,i} = \binom{S_{slot}}{i} \left(P_{n\_smooth,1}\right)^i \left(1 - P_{n\_smooth,1}\right)^{S_{slot}-i} \tag{5-13}$$

If $S_{n\_s\_buffer}$ represents the average number of smoothing buffers for normal playback of a segment of a video, we can write

$$S_{n\_s\_buffer} = \sum_{i=1}^{S_{slot}} \left(P_{n\_smooth,i} \times i \right) \tag{5-14}$$

The probability that a smoothing buffer has been allocated for interactive playback is a joint probability of two independent probabilities. One of them is the probability that there is at least an interactive request, i.e., $\left(1 - e^{-\lambda(1-P_{normal})}\right)$. And the other one is the probability that the interactive client did not join in a running group after resuming normal playback or there is no running group at the time slot of the resuming point, i.e., $\left(e^{-\lambda P_{normal} \times T_{share}}\right)$. Therefore, the probability that a smoothing buffer has been allocated for interactive playback is

$$P_{\text{int}\_smooth,1} = \left(1 - e^{-\lambda(1-P_{normal})}\right) \times \left(e^{-\lambda P_{normal} \times T_{share}}\right) \tag{5-15}$$

Therefore, the probability that i smoothing buffers have been allocated for a segment due to interactive playback is

$$P_{\text{int}\_smooth,i} = \binom{S_{slot}}{i} \left(P_{\text{int}\_smooth,1}\right)^{i} \left(1 - P_{\text{int}\_smooth,1}\right)^{S_{slot}-i} \tag{5-16}$$

If $S_{\text{int}\_s\_buffer}$ represents the average number of smoothing buffers for interactive playback for a segment of a video, we can write

$$S_{\text{int}\_s\_buffer} = \sum_{i=1}^{S_{slot}} \left(P_{\text{int}\_smooth,i} \times i \right) \tag{5-17}$$

The computation of $P_{n\_smooth,1}$, $P_{\text{int}\_smooth,1}$, $S_{n\_s\_buffer}$ and $S_{\text{int}\_s\_buffer}$ is different if $S_{avg,length} = S_{\min,length}$. In this case, the buffer is shared among the clients whose requests fall within an interval equal to $S_{\min,length}$ seconds. Since only one flat smoothing buffer is

allocated for a segment, therefore, $P_{n\_smooth,1}$, $P_{\text{int}\_smooth,1}$, $S_{n\_s\_buffer}$ and $S_{\text{int}\_s\_buffer}$ can be computed as follows:

$$P_{n\_smooth,1} = \left(1 - e^{-\lambda P_{normal} \times S_{\min,length}}\right) \tag{5-18}$$

$$S_{n\_s\_buffer} = P_{n\_smooth,1} \tag{5-19}$$

$$P_{\text{int}\_smooth,1} = \left(1 - e^{-\lambda (1 - P_{normal})}\right) \times \left(e^{-\lambda P_{normal} S_{\min,\, length}}\right) \tag{5-20}$$

$$S_{\text{int}\_s\_buffer} = P_{\text{int}\_smooth,1} \tag{5-21}$$

Adding $S_{n\_s\_buffer}$ and $S_{\text{int}\_s\_buffer}$ we can find the average number of smoothing buffers required for a segment. Multiplying the average number of smoothing buffers for a segment with the average number of segments of a video we can find the total number of smoothing buffers required for a video. Multiplying the total number of smoothing buffers with the smoothing buffer size, we can compute the total amount of smoothing buffer required for a video. Each segment requires a prefix buffer. Multiplying the average number of segments with the prefix buffer size, we can compute the total amount of prefix buffer required for a video. In order to find the total amount of buffer requirement for a video we add prefix and smoothing buffer amounts in equation (5-22). If B represents the total amount of buffer required for a video, we can write

$$B = S_{avg,number} \times T_{prefix}$$
$$+ S_{avg,number} \times \left(S_{n\_s\_buffer} + S_{\text{int}\_s\_buffer}\right) \times T_{smoothing} \tag{5-22}$$

5.6. Server-proxy Network Bandwidth Requirement

In Section 5.3, we have already mentioned that the minimum segment length, i.e., the sharing interval can be optimized for better server-proxy bandwidth management. In this

section, we will find the server-proxy bandwidth requirement for a video in terms of the sharing interval T_{share}. We have assumed that the server is transmitting the video to the proxy at a rate $r_{s,p}$ time less/more than that of the mean-bit-rate, b bits per second (bps), of the video. For each stream the server needs $(r_{s,p} \times b)$ bps of server-proxy network bandwidth. It is obvious that the number of server streams required is equal to the number of smoothing buffers required, i.e., the server needs $S_{avg,number}(S_{n\_s\_buffer} + S_{\text{int}\_s\_buffer})$ streams on the average to stream the video. We can compute the average server-proxy bandwidth requirement χ bps for the video as

$$\chi = S_{avg,number}(S_{n\_s\_buffer} + S_{\text{int}\_s\_buffer}) \times r_{s,p} \times b \qquad (5\text{-}23)$$

5.7. Simulation Results

We have written a simulation program in Java in order to evaluate the performance of our proposed streaming model. We have assumed 100 ms minimum delay and 400 ms delay jitter between the server and the proxy. We have considered one video in our simulation. We obtained the video meta-data from the video trace of a high quality MPEG-4 encoded video (Jurassic Park I) from [80]. This video is one hour (90000 frames) long with a frame rate 25 frames/s. Its mean-bit-rate is 0.77 Mbps. We have computed the sizes of the prefix and the work-ahead smoothing buffers of the non-segmented video using expressions (5-2) and (5-3), respectively. Computed values of the prefix and work-ahead buffers are 15.6032 MB and 15.6013 MB respectively. Dividing these MB values by the mean-bit-rate we get their equivalent time values 162.89 seconds and 162.87 seconds respectively. In our original scheme, proposed in Section 5.2, new prefix buffer size and work-ahead buffer size are computed for each new segment. The use of a fixed prefix buffer size and a fixed work-ahead buffer size for all the segments will not make a big difference in the performance study. For this reason, we have used a fixed prefix buffer size (162.89 seconds) and a fixed work-ahead buffer size (162.87 seconds) in our

simulation in order to make our computation simple. When it is not otherwise stated, we have used several default values to run the simulation. They are as follows:

Average request rate $\lambda = 1$ request/minute

Minimum segment length $S_{\min,length} = 10$ minutes

Server-to-proxy transmission rate ratio $r_{s,p} = 1.0$

Probability of interactive play $(1 - P_{normal}) = 0.5$

In Figure 5.7, we plot the average numbers of new segments that are being created at the proxy during a single round against the request rates. Here, by a round we mean one full-length playback of a video. The average number of new segments in a round also represents the average number of new prefix buffers allocated for the video in a round. Figure 5.7 shows that more new segments are created or more new prefix buffers are allocated for the video when the request rate increases. For this reason, the number of rounds or the total segmentation time required to reach the segmentation saturation point of a video decreases as the request rate increases. This has been shown in Figure 5.8.

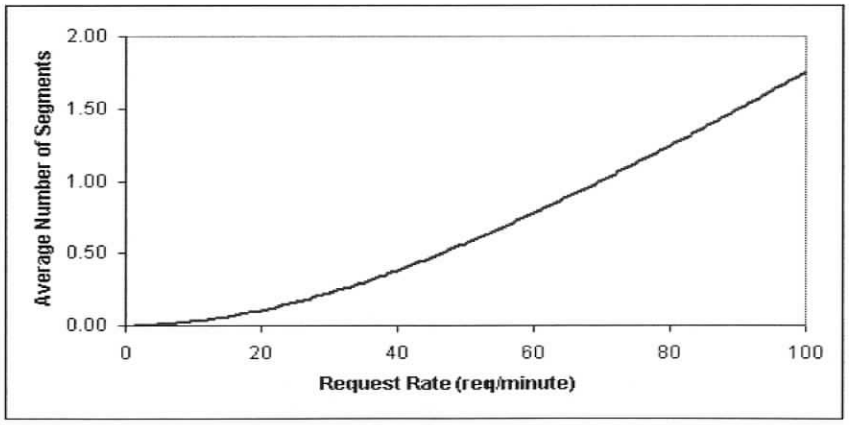

Figure 5.7: Average Number of Segments created in a round vs. Request Rate

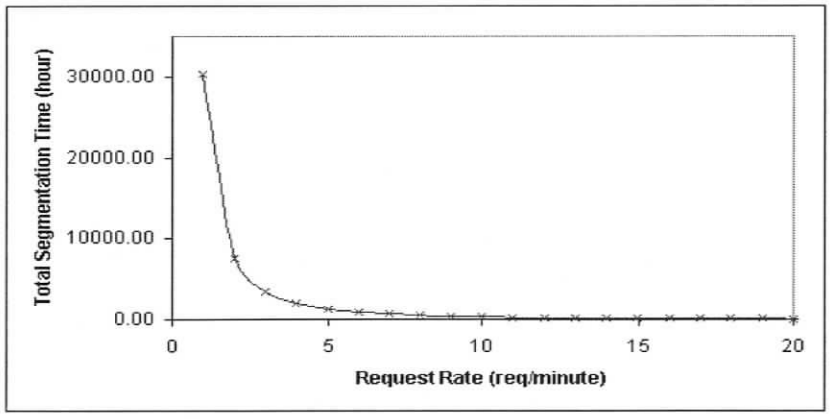

Figure 5.8: Total Segmentation Time with respect to Request Rate

The reason for the segmentation time behaviour in Figure 5.8 is that the number of interactive playbacks increases proportionately as the request rate increases. These interactive playback triggers the segmentation process. Therefore, more segments are created and more prefix buffers are allocated as the number of interactive playback requests increases. For the same reason, more segments are created and more prefix buffers are allocated when the probability of interactive playback increases, which also leads to a smaller segmentation time. These have been shown in Figure 5.9 and Figure 5.10 respectively.

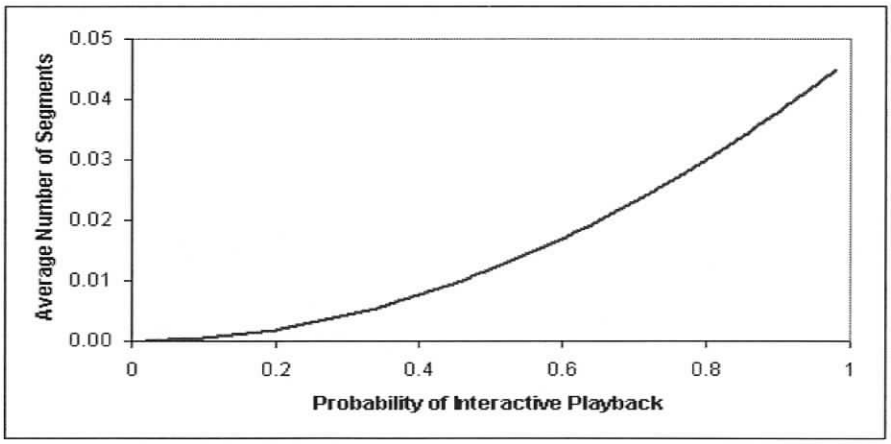

Figure 5.9: Average Number of Segments created in a round with respect to the Probability of Interactive playback

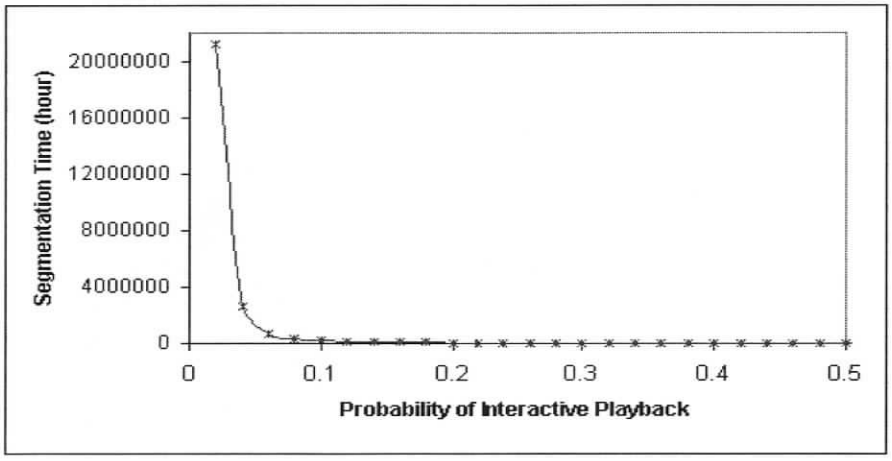

Figure 5.10: Total Segmentation Time with respect to the Probability of Interactive playback

In Section 5.3 we have discussed that the value of the minimum segment length determines the proxy-buffer and the server-proxy bandwidth requirements. Our simulation results confirmed our predictions. It is evident from Figure 5.11 that the server-proxy bandwidth requirement decreases as the value of the minimum segment length increases. This is because for a large value of the minimum-segment-length, smoothing buffers quickly transform into flat buffers. In this case, one server stream for one segment is sufficient to serve all the clients of that segment. Moreover, at a large value of the minimum-segment-length the average number of segments for a video decreases. These two factors together reduce the number of server streams required for a video, i.e., reduces the server-proxy bandwidth requirement.

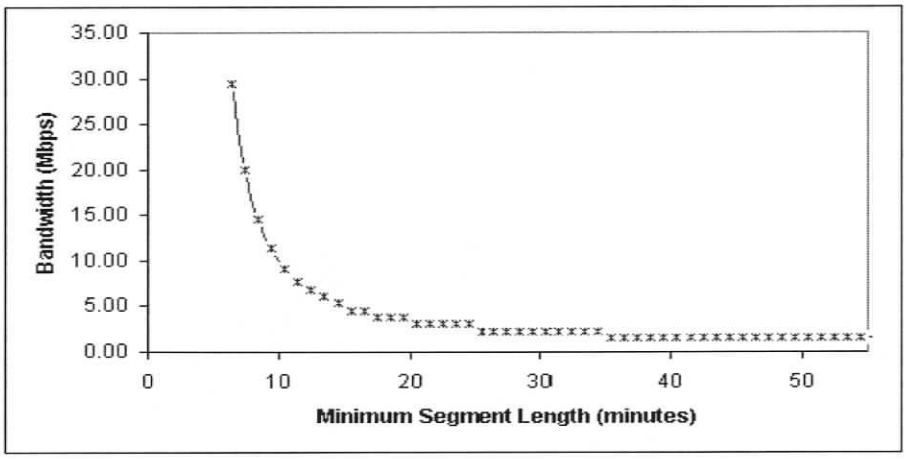

Figure 5.11: Server-proxy bandwidth requirements with respect to Minimum Segment Length.

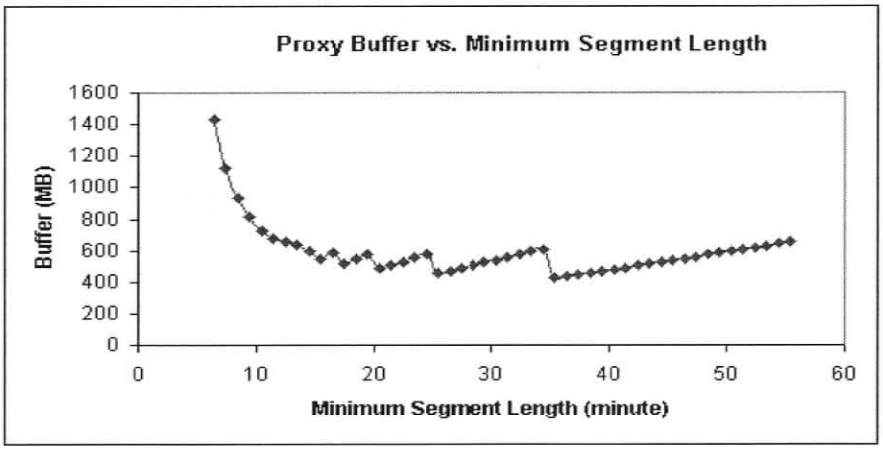

Figure 5.12: Proxy-buffer requirements with respect to Minimum Segment Length

From Figure 5.12 we observe that the proxy buffer (prefix + smoothing) requirement decreases first as the value of the minimum-segment-length increases. It stops decreasing proxy buffer requirement after a certain value (17.5 minutes) of the minimum-segment-length. The above buffer requirement, rather, increases with the minimum-segment-length after that point ([17.5, 516.5] point in Figure 5.12). The requirement decreases at first due to the quick transformation of circular smoothing buffers into flat smoothing buffers and due to the reduction in the number of segment prefixes at a large minimum-

segment-length. Though this transformation happens even quicker with a larger minimum-segment-length, it starts increasing rather than reducing the total buffer requirement. This is because with a larger value of the minimum-segment-length the average size of the smoothing buffer become very large and the average holding time of these large smoothing buffers also become large. Figure 5.12 exhibits some ups and downs in proxy buffer requirement. The reasons for this behaviour are as follows. The amount of smoothing buffer required for interactive playback is proportional to the minimum-segment-length until the average-segment-length is greater than the minimum-segment-length. However, above buffer requirement becomes inversely proportional to the minimum-segment-length when the average-segment-length is equal to the minimum-segment-length. The average-segment-length also grows slower than the minimum-segment-length.

From Figure 5.11 and Figure 5.12 we can conclude that we should not choose either a very small or a very large value for the minimum segment length. Choosing a very large value for the minimum-segment-length also reduces the granularity of the interactive playback modes, which is not desirable. We propose to choose a value between 10 to 15 minutes for the minimum segment length of a video.

5.8. Chapter Summary

We have proposed an on-demand video segmentation and proxy-buffer provisioning scheme for scalable streaming system that can provide interactive video playback modes. It provides interactive playback modes from the start though it starts streaming with a non-segmented video and with a single prefix at the proxy. Our scheme divides a video into many segments and caches the prefixes of the new segments only after receiving interactive playback requests, i.e., segmentation and prefix caching are done on-demand. It results in better resource utilization by skipping pre-segmentation and pre-prefix-caching. By choosing an appropriate value for the minimum-segment-length it can control the server-proxy bandwidth requirement and the proxy-buffer requirement.

Simulation results have shown the effectiveness of our scheme. Our scheme cannot control the network bandwidth requirement on the proxy-client path. A single proxy has limited bandwidth on this path. If the level of demand for both interactive and non-interactive playback of the videos is very high, one proxy might not be able to meet all the demands. We can deploy multiple proxies to meet this high demand. If these proxies remain isolated from each other, the overall resource utilization of the system will be very poor. In the next chapter, we present a scalable streaming scheme where several proxies in a cluster can collaborate to increase their overall resource utilization.

6. Collaborative Proxy Peering System

6.1. Introduction

The present Internet is mainly unicast enabled, in part because deployment of multicasting in the Internet is costly. This is because additional router support is required for multicast (e.g., support for additional functions or maintenance of per-group state information at core/edge routers). The present rate of multicast deployment suggests that multicast enabled IP networks will not be widely available in the near future. Therefore the prospect for multicast-based scalable streaming schemes [62][63] is not encouraging. A proxy based scalable streaming scheme that has been presented in Chapter 3 is an alternative solution over the present unicast-only Internet. A proxy based scalable streaming scheme, like a multicast-channel based scalable streaming scheme, shares a server-proxy stream among many clients, which reduces the load on the server as well as the network bandwidth requirement between the server and the proxy. Each individual client, however, gets its own, individual stream from the proxy, i.e., proxy-client streams are not shared. Figure 6.1 and Figure 6.2 show different stream sharing mechanisms in multicast and proxy based scalable streaming schemes, respectively.

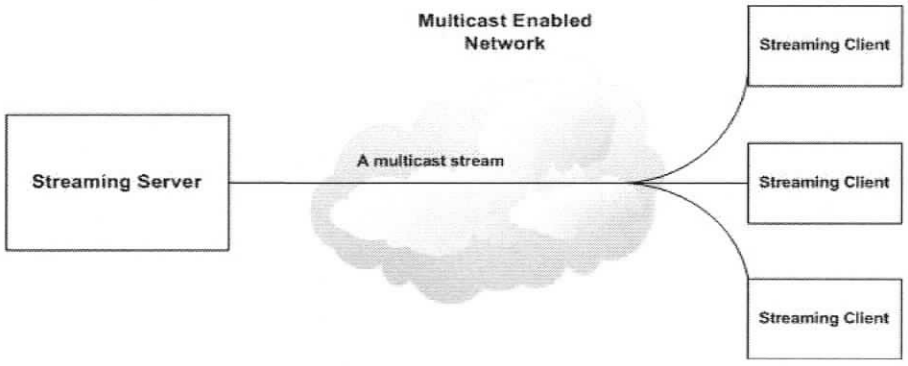

Figure 6.1: Video Stream in a Multicast-based Scalable Streaming Scheme

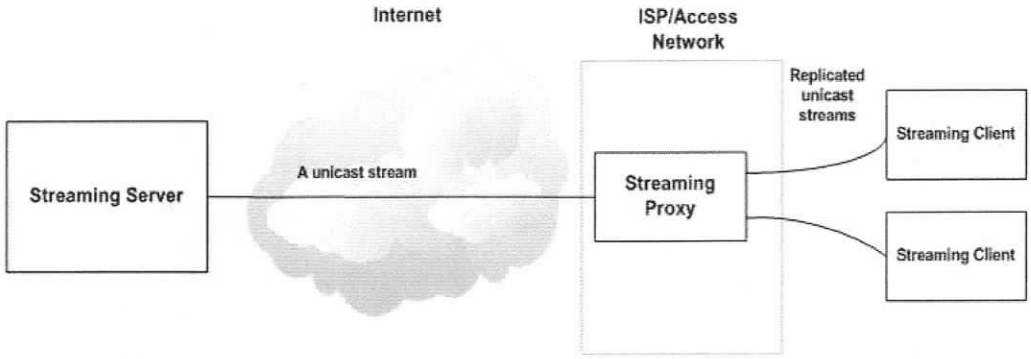

Figure 6.2: Video Stream in a Unicast-based (Proxy-based) Scalable Streaming Scheme

The load on the proxy, as well as the bandwidth requirement on the proxy-client path, increases linearly as the number of clients at a proxy increases in a proxy-based scalable streaming scheme. The performance bottleneck is thus shifted from the server to the proxy. However, a set of proxies can be used in order to overcome the new bottleneck. If the proxies in the set remain isolated, i.e., non-collaborative, overall resource utilization at these proxies will be very poor. It will also lack other benefits that a collaborative proxy-peering system (CPPS) can provide. Figure 6.3 shows two non-collaborative proxies concurrently fetching two streams of the same video from the server. However, two collaborative proxies in a CPPS, in Figure 6.4, fetch only one stream of the same video from the server.

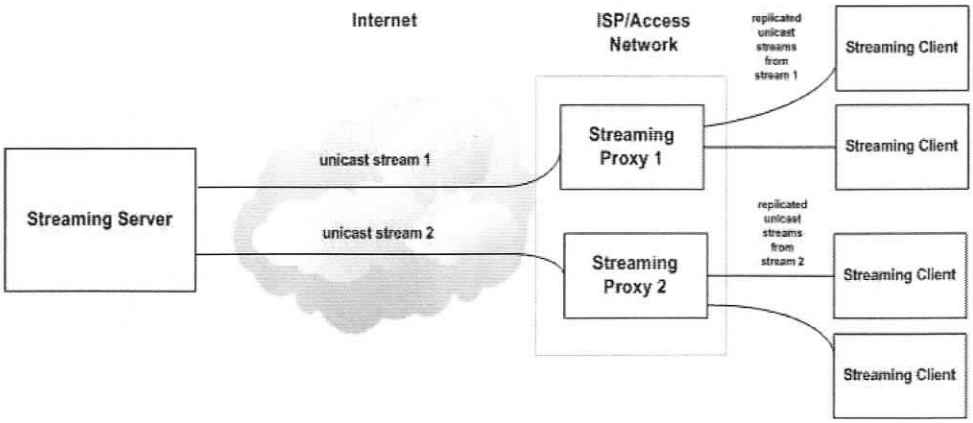

Figure 6.3: Non-cooperative proxies in an ISP/access network

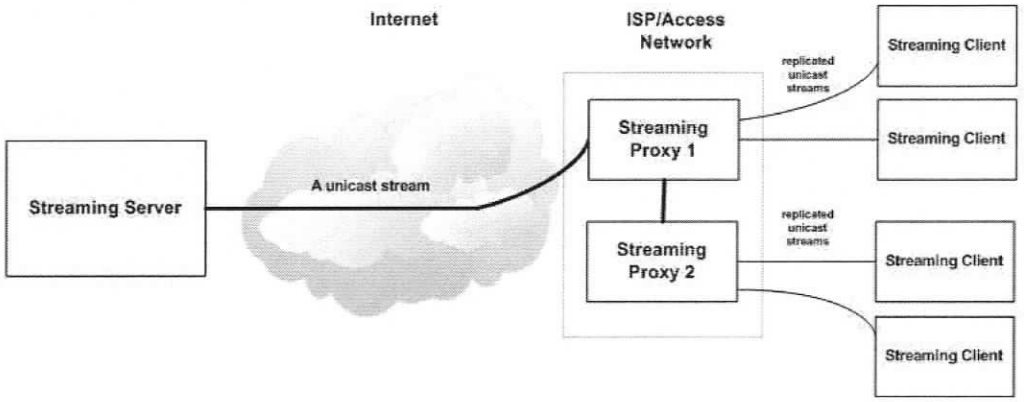

Figure 6.4: Cooperative proxies in a CPPS

In this chapter, we propose that the proxies, that serve a particular video and are in the same ISP, should form a CPPS. Our proposed proxy collaboration reduces the total number of server-proxy streams and the total proxy buffer requirement. It also enhances the performance of on-demand segmentation and buffer provisioning techniques.

The rest of the chapter is organized as follows. We describe related research work in Section 6.2. We present the design of our collaborative proxy peering system in Section 6.3. We compute the performance gain through our CPPS design in Section 6.4. We compute the bandwidth and buffer savings through collaborative proxy peering in Section 6.5. In Section 6.6, we present some results showing the benefits of our proposed CPPS. Finally, we conclude the chapter in Section 6.7.

6.2. Related Works

Acharya et al. [101] proposed a video caching proxy collaboration system called *MiddleMan*. It assumes that a group of cooperative proxies in a Local Area Network (LAN) is used to deliver multimedia streams over the Internet. MiddleMan assumes two types of proxies. One is called a local proxy and is only responsible for answering client requests. The other is called a storage proxy; it does not answer client requests; it just stores media data. A coordinator in the LAN coordinates media data distribution and caching, hence coordination remains centralized. The coordinator keeps track of the files cached by each storage proxy and redirects requests accordingly. The storage proxies do not perform cache management tasks though they cache video files. The coordinator performs cache management tasks for the storage proxies. MiddleMan effectively shares caches at multiple storage proxies, hence reduces the total cache storage requirement at the storage proxies. However, it demands that a LAN, which sharply limits geographical coverage, to connect the cooperative proxies and the coordinator. The centralized coordinator is a bottleneck for system throughput and it also remains as a single point of failure. The idea of keeping two kinds of proxies, one for redirecting streams and the other one for caching media data, adds complexity and unnecessary overheads. No stream sharing among a group of concurrent clients at any level is used in MiddleMan. VBR streams are not smoothed out. And no provision to provide interactive playback modes is made.

Hoffman et al. [102] proposed a collaborative caching technique for streaming multimedia over the Internet called *SOCCER (Self-Organizing Cooperative Caching Architecture)*. It also assumes that a group of cooperative proxies, not necessarily in a single LAN, is used to deliver streaming media over the Internet. They proposed a segmentation scheme, a caching scheme, and a self-organizing cooperative caching architecture. A media object is divided into several uniform length segments. Every k contiguous segments form a chunk. Each chunk is then cached independently using prefix cache allocation and replacement policies. A client's request for a stream is either directed to a proxy or intercepted by a proxy. If the stream is accessed for the first time, the intervening proxy will not find it in its own cache or in any other proxy's cache, hence it sends a request to the server. When the stream data from the server arrives at the intervening proxy, it caches the data chunk by chunk in its static cache and forwards it to the client. The intervening proxy also dynamically caches incoming data in a small circular buffer. Each proxy periodically advertises its ability to deliver segments of media streams to other proxies, using the *Expanding Ring Advertisement (ERA)*. If the same stream is accessed later, the intervening proxy will find the stream partially or fully cached in SOCCER caches. If media data is not cached locally in the intervening proxy, it uses a cost function to select the best cache, hence another proxy cache in SOCCER to retrieve media data. The cost function depends on the distance between the intervening proxy and the target proxy and on the load at the remote proxy. The cost function also depends on the overhead factor of dynamic caching if dynamic caching is involved. In any case, the intervening proxy chooses the minimum cost cache. If it chooses a static cache, it gets as much media data as there is in the selected static cache up to the start of the next available local cache or up to the end of the request. If it chooses a dynamic cache, it gets media data from the dynamic cache up to the end of the clip. SOCCER effectively shares both static and dynamic caches at multiple cooperative proxies, even if these proxies are not connected through a single LAN. However, there are some limitations to the SOCCER design. It degenerates into performing prefix caching of the entire media file in the extreme case. Proxy resource advertisement using ERA needs multicast connectivity among the cooperative proxies, which is not available in the

present Internet. VBR streams are not smoothed out and it does not provide interactive playback modes.

Napster [106] and Gnutella [107] provide lookup operations to find data in a distributed set of peers. Napster uses a centralized index, resulting in a single point of failure. Gnutella floods each query over the whole system, so its communication and processing costs are high. For these reasons, we consider neither Napster nor Gnutella suitable for our collaborative proxy peering system.

Stoica et al. [103] have proposed *Chord*, a suite of distributed lookup protocols that is efficient in storing and retrieving data in a peer-to-peer system. Chord associates a key with each data item and produces a key identifier using a consistent hash function SHA-1 [104]. It also creates a peering node identifier by hashing the node's IP address and then maps the key onto a node. Chord implements data location by storing the key/data pair at the node, to which the key maps. We find Chord is a suitable suite of protocols for looking up video data in our collaborative proxy peering system. The Freenet peer-to-peer storage system [105], like Chord, is decentralized and symmetric, and automatically adapts when nodes leave and join. However, we didn't adopt Freenet since its lookup operation does not run in predictable time and it does not always result in success or definitive failure, unlike Chord. Chord's [103] placement protocol can be used to distribute videos among a set of proxies in an ISP and its lookup protocol can be used to retrieve a video from these proxies. Chord allows a proxy to join and leave the set freely and adapts the change efficiently. If a new proxy joins in the set, Chord can assign some videos to it reducing the load from the other proxies. If a proxy leaves the set, Chord can reassign the videos, held by the leaving proxy, to the remaining proxies, i.e., it can redistribute the load. We describe Chord protocols in the Appendix in Section A.7.

6.3. Proposed Collaborative Proxy Peering System

In this section we describe the design of our collaborative proxy peering system (CPPS) using Chord protocols. The maximum number of clients that a single proxy can serve may be much smaller than the total number of clients of a video in an ISP or access network. Multiple proxies can be used to serve this large number of clients. We assume the proxy-proxy links to be fast and reliable and also the overhead to fetch data by one proxy from another proxy in the same ISP to be negligible. We propose to form a CPPS when multiple proxies are used to serve a single video, one CPPS for each video. Chord placement and lookup protocols [103] can be used to store and retrieve videos to and from these proxies. There will be a single Chord ring for the whole ISP, and multiple CPPSs for multiple videos in the ISP will be implemented using this single Chord ring. Chord allows a node, i.e., a proxy, to join and leave the Chord ring freely and adapts the change efficiently. If a new proxy joins in the ring, Chord can assign some videos to it reducing the load from the other proxies. If a proxy leaves the group, Chord can reassign the videos that were held by the leaving proxy to the remaining proxies in the ring, i.e., can redistribute the load.

We propose to start a CPPS for each video with a single pair of proxies. One proxy in the pair is called the *designated proxy* and the other is called the *backup proxy* for the video. m-bit identification numbers are used for the proxies, videos, and the video segments. Each proxy has a unique 32-bit IP address. $m \geq 32$ is chosen in order to avoid proxy identification conflict using consistent hashing. Each video or video segment has a unique Universal Resource Identifier (URI) [111]. An identification number is assigned to a proxy by hashing its IP address and an identification number is assigned to a video by hashing its URI. A proxy is chosen as the designated proxy for a video when Chord placement protocol maps the *id* of the video onto the *ID* of the proxy. A proxy s has been defined as the *successor proxy* of a proxy n if the *ID* of proxy s immediately follows the *ID* of proxy n in the Chord ring. The successor proxy of the designated proxy is chosen as its backup proxy. The designated proxy should cache the base sub-stream and the very

first prefix of the video. The designated proxy should backup video data and the corresponding meta-data in the backup proxy.

Any proxy in the Chord ring can receive a client request for any video. It will forward a client request for a particular video to its designated proxy. A proxy finds the designated proxy of a video by searching through its finger table. A finger table in each proxy maintains the information about the other proxies in the Chord ring. The designated proxy of a video finally receives all its requests and either streams the video itself or redirects a request to an appropriate proxy in its CPPS. It keeps track of the number of proxies in its CPPS; which segment of the related video has been placed in each of these proxies, as well as the number of clients for each segment, i.e., the load on the proxies. After getting a client request the designated proxy forwards it to a proxy in its CPPS, which holds the requested segment and is lightly loaded. The forwarded proxy transmits the video segment to the client. This proxy uses work-ahead smoothing technique, presented in Chapter 3, in order to smooth out the VBR traffic. Also, scaling of streaming is realized by sharing the same server-proxy stream among a group of clients arriving in the same interval using the techniques presented in Chapter 3. Interactive playback modes are provided using the technique described in Chapter 4. On-demand segmentation and buffer provisioning algorithm presented in Chapter 5 is used in parallel in order to create new segments and to cache the new segment prefixes. If the designated proxy fails, the backup proxy takes up the roles of the designated proxy and chooses its successor proxy as the new backup proxy for the video.

We propose to run an instance of the distributed Chord Coordinator Program (CCP) in each proxy in the Chord ring, in order to keep track of the total number of client requests, the total number of accepted requests, and the total capacity of the proxies in the Chord ring. A new proxy will be added to the Chord ring when the total number of client requests exceeds the current capacity and a proxy will be removed from the Chord ring when the total number of client requests is substantially below the current capacity.

When the number of the clients exceeds the total capacity of the designated and the backup proxies, the designated proxy adds a lightly loaded proxy from the Chord ring to its CPPS. The designated proxy continues to add a new proxy in its CPPS until the total number of clients remaining falls below the total capacity. If it fails to add a proxy from the existing Chord ring it ask CCP to add a new proxy in the Chord ring. It distributes the video segments among the proxies in its CPPS and keeps records about the segment distributions. The designated proxy of a video will run a segment distribution-redistribution program when it adds or removes a proxy to or from its CPPS. Developing an algorithm for the segment distribution-redistribution program is the topic of our future research.

A video segment may be assigned to several proxies. When a video segment is assigned to a proxy, i.e., the segment prefix is cached in the proxy, the proxy becomes responsible to stream that segment to the clients. Client requests for that particular video segment will be forwarded to that particular proxy by the designated proxy. Each proxy in a CPPS will also run an on-demand segmentation and buffer provisioning algorithm from Chapter 5 in order to create new segments. The instance of the on-demand segmentation and buffer provisioning algorithm at each proxy in the CPPS will consider all the segments that have already been created in the group while creating a new segment. Any proxy can create a new segment if the segmentation algorithm permits. However, the designated proxy will decide the placement of the new segment prefix using segment distribution and re-distribution algorithm. It may select a different proxy to cache the new segment prefix. The new segment prefix is transferred from its creator proxy to the selected proxy. After receiving the segment prefix, the selected proxy will cache it and report this segment information to the designated proxy.

The designated proxy of the video finds the set of proxies that will be involved in the requested playback. A client needs to switch from one proxy to another proxy in order to

access consecutive segments of the same video when they are distributed among several proxies. The above proxy switching has to be synchronized in order to provide uninterrupted playback. Developing a proxy synchronization algorithm is also the topic of our future research. Designated proxy sends a playback request to each proxy in the set at an appropriate time according to proxy synchronization algorithm.

After receiving a request for a video segment each proxy starts the playback of the segment. Each proxy also keeps track of the number of clients of each video segment are being served. If the number of clients for a particular segment of a video becomes zero, the proxy removes the segment and reports this removal information to the designated proxy. If there is no segment of a video left, the proxy removes itself from the corresponding CPPS. When a proxy detects that it is not a part of any CPPS it asks the CCP to remove it from the Chord ring. If a designated proxy detects that there is no client for the corresponding video it asks the CCP to remove the corresponding CPPS from the ring. The pseudo codes for different algorithms used by our collaborative proxy peering system are outlined below:

```
/* proxy_receive_request algorithm is used by a proxy after receiving a client request. It accepts the reference of a
video and the list of segments of that video to be played. It redirects the request to the designated proxy of the video.
*/
proxy_receive_request (video, list-of-segments) {
    Search for the designated proxy of the video using Chord lookup through the Finger Table
        if (no designated-proxy for the video) {
                while (ccp is not able to create a new cpps in the Chord ring) {
                        Ask ccp to add a new proxy in the Chord ring
                        }
                Create designated and backup proxies for the new cpps of the video
                Distribute segments in the list-of-segments among the designated and backup proxies using
                segment distribution-redistribution algorithm
                }
    Get the designated-proxy of the video
    Redirect the request to the designated proxy
    }
```

```
/* designated_receive_request algorithm is used only by the designated proxy of a video after receiving redirected
request from any other proxy. It finds all the proxies that will be involve in the playback of the video and sends the
playback requests to those proxies.
*/
designated_receive_request (video, list-of-segments) {
        Get the list of proxies corresponds to the list-of-segments
        for each proxy in the list do {
                if (available bandwidth is not enough) {
                        while (a new proxy from the Chord ring cannot be added to the cpps) {
                                Ask ccp to add a new proxy in the Chord ring
                                }
                        Ask the new proxy to get the segment of the video
                        Add the new proxy in the cpps and in the list of the proxies
                        Remove the old proxy from the list of the proxies
                        }
                }
        Synchronize all the proxies in the list to provide continuous playback of the video and ask the proxies to start
        segment playback at an appropriate time according to synchronization algorithm
        }

/* proxy_play_segment algorithm is used by a proxy which holds the requested video segment and have enough
bandwidth to serve the request.
*/
proxy_play_segment (video, segment) {
        Start the playback of the video segment
        if (playback done) {
                if (no more client for the segment) {
                        Remove the segment from the proxy
                        if (no more segment of the video) {
                                Ask designated to remove the proxy from the cpps
                                if (no more video) {
                                        if (this not a designated or not a backup proxy) {
                                                Ask ccp to remove this proxy from the ring
                                        else if (no more client for the corresponding cpps) {
                                                Ask ccp to remove the cpps from the ring
                                                Ask ccp to remove this proxy from the ring
```

```
                                        }
                                    }
                                }
                            }
                        }
                    }
```

/* **proxy_new_segments** algorithm is used by a proxy when it creates new segments. Two new segments are generally created from an current-segment. The references of the video, the old, and the new segments are passed as the parameters of the algorithm.

*/

proxy_new_segments (video, current-segment, new-left-segment, new-right-segment) {

Get the designated proxy of the video using Chord lookup through the Finger Table

Ask the designated proxy to place new segments at the proxies in its cpps using segment distribution-redistribution algorithm

Ask the designated proxy to remove the current-segment from the cpps

}

Implementation details about the design of our CPPS outlined above is beyond the scope of this thesis, and is also a topic for our future research. We complete this chapter showing the benefits of forming a collaborative proxy peering system. We compute the performance gain in on-demand segmentation and buffer provisioning mechanisms in our CPPS in the next section and the bandwidth and buffer savings in our CPPS in Section 6.5.

6.4. Performance gain through Proxy-collaboration

In Section 6.3, we have assumed that the maximum number of concurrent clients that a single proxy can serve is limited. Let this maximum number be Q_q and let the request rate for an L-second video be λ requests/s. The total number of concurrent clients of the video in an ISP will be λL. Therefore, the ISP needs q proxies to serve the video, where

$q = \left\lceil \frac{\lambda L}{Q_q} \right\rceil$, and each proxy will receive $\lambda_q = \frac{\lambda}{q}$ requests per second. In Chapter 5, we have computed the average segmentation time S_{time} that is necessary to create the possible maximum number of segments for a video as the function of its request rate as follows:

$$S_{time} = \left(\frac{S_{\max,number} - 1}{S_{avg,number} - 1} \right) \times L \tag{6-1}$$

Here, $S_{\max,number}$ is the *maximum* possible *number of segments* for a video with duration L seconds and $S_{avg,number}$ is the *average number of segments* that have been created for the video in L seconds. We have computed $S_{\max,number}$ as follows:

$$S_{\max,number} = \left\lceil \frac{L}{S_{\min,length}} \right\rceil \tag{6-2}$$

We have assumed that there are λ requests per second for the video and the probability that a request is for normal playback is P_{normal}. Assuming that there are m video frames in a minimum-segment-length segment and n video frames in the whole video we computed $S_{avg,number}$ as follows:

$$S_{avg,number} = 1 + \begin{cases} \sum_{s=1}^{S_{\max,number}-1} (P_{seg,s} \times s) & if \left\lfloor ((1 - P_{normal})\lambda L) \right\rfloor \geq (S_{\max,Number} - 1) \\ \sum_{s=1}^{\lfloor (1-P_{normal})\lambda L \rfloor} (P_{seg,s} \times s) & otherwise \end{cases}$$

$$P_{seg,s} = \binom{\left\lfloor ((1 - P_{normal})\lambda L) \right\rfloor}{s} (P_{seg,1})^s (1 - P_{seg,1})^{\left\lfloor ((1-P_{normal})\lambda L) \right\rfloor - s}$$

$$P_{seg,1} = \left(\frac{m}{n} \right) \times \left(1 - \frac{m}{n} \right) \times \left(1 - e^{-(1-P_{normal})\lambda} - (1 - P_{normal})\lambda e^{1(1-P_{normal})\lambda} \right) \tag{6-3}$$

If the proxies remain non-collaborative, each proxy runs the segmentation algorithm independently. Each proxy then considers the segments that have already been created only by itself while creating a new segment. It does not consider the segments that have already been created by other proxies for the same video. For this reason, we need to compute $S_{time,non-collborative}$ for each proxy using λ_q as the request rate. If the proxies form a collaborative proxy peering system as proposed in Section 6.3, each proxy runs the segmentation algorithm parallel to other proxies in the peering system. Each proxy considers all the segments that have already been created by all the proxies in the peering system while creating a new segment. Therefore, we can compute $S_{time,collborative}$ for the peering system using λ as the request rate. Collaborative proxy peering system will obviously create more segments in a single round and its segmentation time will be less than that of non-collaborative proxies. We define the difference between $S_{time,non-collborative}$ and $S_{time,collborative}$ as the *segmentation-time-gain* of a collaborative proxy peering system. We present our simulation result showing above segmentation-time-gain in Section 6.6.

6.5. Bandwidth and Buffer Savings through Proxy-collaboration

If the proxies remain non-collaborative they cannot share the segment prefixes with each other and each proxy needs to cache the segment prefixes separately. The maximum number of segments that can be created for a video is $S_{\max,numer}$ and there are q non-collaborative proxies in an ISP, therefore, the ISP will need $q \times S_{\max,number}$ segment prefixes in total. If q proxies in an ISP form a collaborative proxy peering system as proposed in Section 6.3, they can share segment prefixes from each other and the ISP will need only $S_{\max,numer}$ segment prefixes in total. This is a great saving of proxy buffer in a collaborative proxy peering system.

In Chapter 5, we have also computed smoothing buffer requirement B at the proxy and server-proxy bandwidth requirement χ for a video as the functions of its request rate as follows:

$$B = S_{avg,number} \times T_{prefix} + S_{avg,number} \times \left(S_{n\_s\_buffer} + S_{\text{int}\_s\_buffer}\right) \times T_{smoothing}$$

$$\chi = \left(S_{avg,number} \times \left(S_{n\_s\_buffer} + S_{\text{int}\_s\_buffer}\right)\right) \times r_{s,p} \times b \qquad (6\text{-}4)$$

Here, T_{prefix} and $T_{smoothing}$ are the sizes of the prefix and the smoothing buffers in seconds, respectively. $S_{n\_s\_buffer}$ and $S_{\text{int}\_s\_buffer}$ are the numbers of smoothing buffers required for a video at the proxy for normal and interactive playback respectively. $r_{s,p}$ is the rate ratio between the transmission rate and the mean-encoding-bit rate, b bits/s, of the video. We refer to Chapter 3 to see the detail computation of T_{prefix} and $T_{smoothing}$ and to Chapter 5 to see the detail computation of $S_{n\_s\_buffer}$ and $S_{\text{int}\_s\_buffer}$.

If the proxies remain non-collaborative, they cannot share smoothing buffers with each other. For this reason, we need to compute the smoothing buffer requirement B_q and the server-proxy bandwidth requirement χ_q at each proxy using λ_q as the request rate. Finally, we can compute the total smoothing buffer requirement $B_{isp,non-collaborative}$ and the total server-proxy bandwidth requirement $\chi_{isp,non-collaborative}$ for the ISP by multiplying B_q and χ_q by q respectively, i.e.,

$$B_{isp,non\_collaborative} = q \times B_q \qquad (6\text{-}5)$$

$$\chi_{isp,non-collaborative} = q \times \chi_q \qquad (6\text{-}6)$$

If the proxies form a proposed collaborative peering system they can share smoothing buffers with each other and we can compute the total smoothing buffer requirement

$B_{isp,collaborative}$ and the total server-proxy bandwidth requirement $\chi_{isp,collaborative}$ for the ISP using λ as the request rate. Both $B_{isp,collaborative}$ and $\chi_{isp,collaborative}$ will be less than $B_{isp,non-collaborative}$ and $\chi_{isp,non-collaborative}$ respectively. Therefore, the proxy collaboration saves server-proxy bandwidth and it saves not only the segments prefix buffer but also the smoothing buffer of an ISP. In Section 6.6, we present simulation results showing above bandwidth and buffer savings of a collaborative proxy peering system.

6.6. Simulation Results

We extended the Java program used in the previous chapter in order to demonstrate the performance gain and the savings in server-proxy bandwidth and in proxy buffer space by our proposed proxy peering system. Like Chapter 5, we have assumed 100 ms minimum delay and 400 ms maximum delay jitter between the server and the proxy. We have also considered one video in our simulation. We have used the same video meta-data from the video trace of a high quality MPEG-4 encoded video (Jurassic Park I) from [80]. We have also used some default values to run the simulation. They are as follows:

Minimum segment length $S_{\min,length} = 5$ minutes

Server-to-proxy transmission rate factor $r_{s,p} = 1.0$

Probability of interactive play $(1 - P_{normal}) = 0.5$

The mean encoding bits rate of the video is 0.77 Mbps. We have assumed that each proxy has OC-3 bandwidth 155.5 Mbps and can serve the video to approximately 200 concurrent clients, i.e., $C_q = 200$. The duration of our video is 1 hour, i.e. $L = 60 \times 60$ seconds. In order to make our computation simple, we have assumed that the request rate for a video λ can increase only at a multiple of $\frac{C_q}{L}$, i.e., $\frac{200}{60 \times 60}$. In this case, a newly joined proxy always receives the number of requests at its full capacity.

We have computed segmentation time gain by the proxy peering system by subtracting $S_{time,collborative}$ from $S_{time,non-collborative}$. We plot these segmentation time gains against the number of proxies in the peering system in Figure 6.5. More proxies join in the peering system in order to meet the increasing request rates. An increase in the request rate proportionately increases the number of interactive requests, i.e., the number of segmentation operations. In the proxy peering system all the proxies together create the segments of a video and reach the possible maximum number of segments quickly. For this reason, segmentation time with a larger number of proxies in the peering system becomes shorter. In a non-collaborative proxy system proxies create segments individually. Only its own interactive requests trigger segmentation operations in a proxy, i.e., few segments are created in a given time in a proxy. Each proxy alone needs a long time to create the maximum possible number of segments. For this reason, segmentation time even with a large number of proxies in a non-collaborative system remains long and fixed. Therefore, segmentation time gain increases as the number of proxies in proxy peering system increases and that is shown in Figure 6.5.

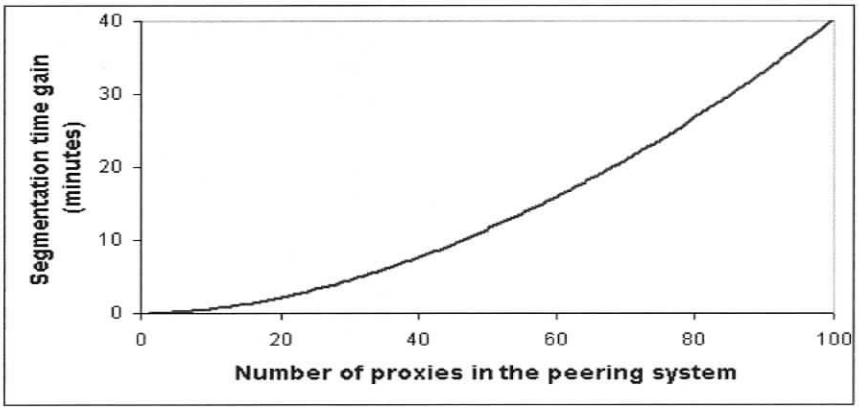

Figure 6.5: Segmentation time gain vs. Number of Proxies in a peering system.

We have computed server-proxy bandwidth savings in the proxy peering system by subtracting $\chi_{isp,collaborative}$ from $\chi_{isp,non-collaborative}$. We plot these server-proxy bandwidth

savings against the number of proxies in the peering system in logarithmic scale in Figure 6.6.

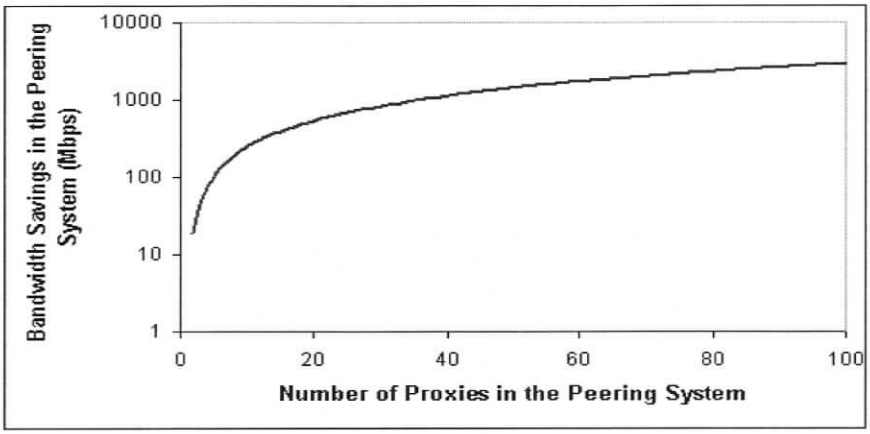

Figure 6.6: Server-proxy bandwidth savings vs. Number of Proxies in a peering system.

We have also computed smoothing buffer savings at the proxies in the proxy peering system by subtracting $B_{isp,collaborative}$ from $B_{isp,non-collaborative}$. We plot these smoothing buffer savings against the number of proxies in the peering system in logarithmic scale in Figure 6.7.

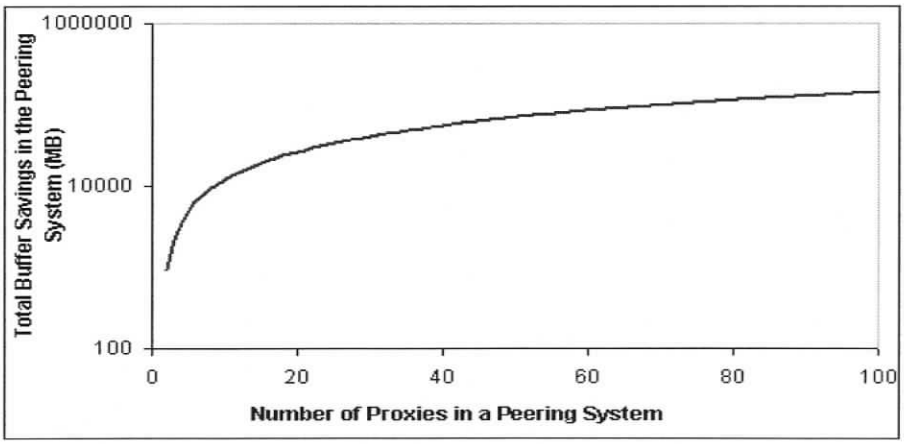

Figure 6.7: Smoothing buffer savings vs. Number of Proxies in a peering system.

Figure 6.6 and Figure 6.7 show that the server-proxy bandwidth saving and the smoothing buffer saving increase as more proxies join in the peering system, respectively. In the proxy peering system proxies share smoothing buffers and server-proxy streams with each other. As the number of collaborating proxies increases so does this sharing, which increases both server-proxy bandwidth and smoothing buffer savings.

6.7. Chapter Summary

We have designed a collaborative proxy peering system or CPPS for scalable and interactive streaming over the Internet. We have shown that our CPPS achieves improved performance in on-demand segmentation and buffer provisioning while serving a very large number of interactive clients. It also saves substantial amounts of server-proxy bandwidth and proxy buffer space at an ISP.

We have proposed to use the Chord protocols [103] as a suitable suite of lookup and placement protocols for our CPPS. In our future research, we will look at the implementation of our CPPS using Chord protocols in details.

7. Conclusion

We have described our research work in detail in Chapters 3, 4, 5, and 6. In this chapter, we summarize our novel contributions and planned future work.

7.1. Contributions

In this dissertation, we have proposed and analyzed a CBR-transmission scheme for VBR-encoded videos, a scalable streaming scheme, an interactive streaming scheme, an on-demand video segmentation and proxy buffer provisioning mechanism, and a collaborative proxy peering system. In this section, we summarize our contributions for each of the above schemes or systems.

7.1.1 CBR-transmission Scheme for VBR encoded Videos

In Chapter 3, we have presented a proxy based CBR-streaming scheme for VBR-encoded stored videos over the Internet. The novel features of our CBR-transmission scheme are as follows:

- It eliminates the streaming rate variability completely, i.e., enables CBR streaming of VBR-encoded videos.
- It reduces the peak bandwidth requirement close to the mean bandwidth requirement both on the proxy-client path but and on the server-proxy path.
- It is relatively easy to implement.

7.1.2 Scalable Streaming Scheme for VBR encoded videos

In Chapter 3, we have presented a proxy based scalable streaming scheme for both CBR- and VBR-encoded stored videos over the Internet. The novel features of our scalable streaming scheme are as follows.

- Our scheme always remains scalable, sharing a server stream among many clients
- Our CBR-transmission scheme reduces the bandwidth requirement for each stream on both the server-proxy and on the proxy-client paths, i.e., it enables a streaming server and a proxy to transmit more streams concurrently, which further increases the system scalability.
- By sharing the same server stream among many clients through the smoothing buffer, our scheme remains scalable and effective for both CBR- and VBR-encoded videos.
- It eliminates the jitter free, just-in-time arrival requirement on the network
- The client requirements remain minimal, i.e., a client with only a single channel and a small buffer can share a server stream with many other clients.

7.1.3 Scalable and Interactive Streaming Scheme for VBR encoded videos

In Chapter 4, we have presented a proxy-based scalable and interactive streaming scheme for use over the Internet. We use video segmentation and hybrid temporal-data-partition scalable video encoding to create a suitable playback sequence for interactive playback modes. A video is encoded into two sub-streams: base and enhancement sub-streams. The base sub-stream, which is a low quality sub-stream of moderate size, is cached at the proxy in order to play it in the fast forward and backward playback modes. The novel features of our scalable and interactive streaming scheme are as follows:

- It is scalable, uses a proxy instead of a multicast channel and supports the entire set of generally required interactive playback modes from the proxy, without losing its scalability.

- The proxy - without any server intervention - supports all the usual interactive playback modes.
- It uses only one stream composed of two sub-streams and does not need dedicated interactive server stream to support client interaction.
- It does not need a large client buffer or complicated processing at each client.
- It does not increase server-proxy bandwidth or proxy buffer requirements as the level of interaction increases. It is able to serve any large number of clients using a certain part of the available server-proxy network bandwidth and proxy buffers.

7.1.4 On-demand Video Segmentation and Buffer Provisioning Scheme

In Chapter 5, we have presented a simple on demand segmentation and proxy-buffer provisioning scheme. The followings are the main and novel features of our on-demand segmentation and buffer provisioning scheme.

- It avoids over provisioning of proxy buffer space for a video.
- It gradually divides a video into many segments and allocates buffers for these new segments as the number of interactive requests increases.
- Segments are generated using clients' playback behaviours.
- It does not need to run complex scene/shot/frame detection algorithms for segmentation. It does not need to divide segments blindly based on media time.

7.1.5 Collaborative-Proxy-Peering System Design

In Chapter 6, we have presented the design of a collaborative-proxy-peering system (CPPS). The proxies, who serve a particular video and are in the same ISP, should be grouped to form a CPPS. In our CPPS design, we use the Chord [103] placement protocol

to place videos or video segments onto the proxies and its lookup protocol to retrieve videos or video segments from the proxies. The main features of our CPPS are as follows:

- It reduces the total number of server-proxy streams and the total proxy buffer requirement.
- It also reduces the segmentation time of on-demand segmentation techniques.

7.2. Future Work

Our current research work has led us to some issues that need further research. In this section, we list some of them.

- ***Scalable Streaming using Multicast enabled Network***s: We have assumed that the connection between a server and a proxy is only unicast enabled. This is because multicast routing is not widely available in the present Internet and its deployment rate suggests that it will not be generally available in the near future. However, multicast enabled networks might be available in the long term. In that case, further research will be required to take the advantage of a universally multicast-enabled Internet.

  We have also assumed that the local loop or last-mile connection between a proxy and a client is only unicast enabled. Though this is the case for current DSL connections and early Cable Network (DOCSIS) connections, multicast is being implemented in cable networks today. Further research is required in order to take the advantage of the multicast enabled cable networks.

- ***Scalable Streaming of Live Media***: Our scalable streaming scheme is designed for streaming stored videos. In many applications, e.g., in live sports events casting, it is necessary to stream live videos directly from the cameras.

Our CBR-transmission scheme uses video-meta-data, such as the total number of frames, their sizes, and their decoding order to do work-ahead smoothing. The complete set of meta-data of a live stream is not often available at the beginning of the streaming. Further research is required to perform smoothing operation on live streaming in order to make it scalable and economical over the Internet.

- ***Scalable Streaming using WiFi and WiMAX***: Wireless access technologies like WiFi (802.11) and WiMAX (802.16) have recently emerged as promising competitors to the already established wired broadband access technologies such as DSL and cable. Reference [113] describes work to deliver Hi-Quality MPEG-2 Video over Fixed Wireless Access Technologies (802.11b). Further research is required to combine their work with our work to make the streaming scheme scalable using WiFi and WiMAX access technologies.

- ***Optimum Bandwidth and Buffer Allocation***: The bandwidth and memory available at a proxy may be limited. If a proxy has to stream a large number of videos to a large number of asynchronous clients then the bandwidth and the buffer requirements at the proxy may exceed its capacity. In that case, some clients need to be refused or the streaming of some videos needs to be refused. Further research is required to perform the above admission control task at the proxies. The available bandwidth and memory may be allocated to the admitted videos based on their popularity, duration, net revenue realized or other parameters. It is obvious that further research is necessary to find an optimum allocation algorithm.

- ***Segment distribution and redistribution algorithms for our proposed collaborative-proxy-peering system or CPPS***: The designated proxy of a video needs to distribute and redistribute the video segments among the proxies in its

CPPS when a new proxy is added to or removed from the CPPS, when a new proxy is added or remove to or from the Chord ring, or when a new video segment is created for the video. In this thesis, we did not develop a segment distribution-redistribution algorithm. More research is required to develop such segment distribution and redistribution algorithms.

- ***Proxy synchronization algorithm for our proposed CPPS***: The various segments of a video are distributed among the proxies in our CPPS. A client switches from one proxy to another proxy in order to access the consecutive video segments of the video from different proxies. In this thesis, we did not develop the proxy synchronization algorithm, which is required to provide uninterrupted video playback during proxy switching. More research is needed to develop such a proxy synchronization algorithm.

8. Bibliography

[1] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," IETF, RFC-1889, Jan 1996. [Online]. Available: http://www.faqs.org/rfcs/rfc1889.html.

[2] H. Schulzrinne, A. Rao, R. Lanphier, "Real Time Streaming Protocol (RTSP)," IETF, RFC-2326, Apr 1998. [Online]. Available: http://www.faqs.org/rfcs/rfc2326.html.

[3] Internet Engineering Task Force (IETF). [Online]. Available: http://www.ietf.org/.

[4] R. Braden, D. Clark, and S. Shenker, "Integrated Services in the Internet Architecture: An Overview," IETF, RFC 1633, July 1994. [Online]. Available: http://www.faqs.org/rfcs/rfc1633.html.

[5] M. Carlson, W. Weiss, S. Blake, Z. Wang, D. Black, and E. Davies, "An Architecture for Differentiated Services," IETF, RFC 2475, December 1998. [Online]. Available: http://www.faqs.org/rfcs/rfc2475.html.

[6] E. Rosen, A. Viswanathan, R. Callon, "Multiprotocol Label Switching Architecture," IETF, RFC 3031. [Online]. Available: http://www.faqs.org/rfcs/rfc3031.html.

[7] D. O. Awduche, Y. Rekhter, J. Drake, R. Coltun, "Multi-Protocol Lambda Switching: Combining MPLS Traffic Engineering Control With Optical Crossconnects," IETF draft work in progress, April 2001.

[8] R. Fielding, J. Gettys, J. Mogul, H. Nielsen, and T. Berners-Lee, "HTTP/1.1: Hypertext transfer protocol," IETF, RFC 2616, June 1999. [Online]. Available: http://www.faqs.org/rfcs/rfc2616.html.

[9] M. Day, B. Cain, G. Tomlinson, and P. Rzewski, "A Model for Content Internetworking (CDI)," IETF, RFC 3466, February 2003. [Online]. Available: http://www.faqs.org/rfcs/rfc3466.html.

[10] Video Codec for Audiovisual Services at p×64 bits/sec, ITU-T Recommendation H.261, version 1 (Dec 1990), version 2 (1993).

[11] D. J. LeGall, "MPEG: A Video Compression Standard for Multimedia Applications," *Communications of the ACM*, vol. 34, no. 4, pp. 46-58, 1991.

[12] B. G. Haskell, A. Puri, and A. Netravali, "*Digital Video: An Introduction to MPEG-2*, New York: Chapman & Hall, 1997.

[13] Information Technology-Generic Coding of Moving Pictures and Associated Audio Information, International Standard: ISO/IEC 13818, Parts 1-10, 1994.

[14] A. Puri and A. Eleftheriadis, "MPEG-4: An object-based multimedia coding standard supporting mobile applications", *ACM, Journal of Mobile Network and Applications*, pp. 5-32, August 1998.

[15] Information Technology-Generic Coding of Audio-Visual Objects, International Standard: ISO/IEC 14496, Parts 1-6, 1998.

[16] I. Dalgic and F. Tobagi, "Constant Quality Video Encoding", *IEEE Proceeding ICC'95*, Seattle, Washington, pp. 1-7, 1995.

[17] S. Sen, J. Rexford, and D. Towsley, Proxy Prefix Caching for Multimedia Streams, i*n Proc. of IEEE INFOCOM*, New York, vol. 3, pp. 1310-1319, March 1999.

[18] B. Wang, S. Sen, M. Adler and D. Towsley, Optimal Proxy Cache Allocation for Efficient Streaming Media Distribution, i*n Proc. of IEEE INFOCOM*, pp. 366-374, 2002.

[19] L. Boroczky, A. Y. Ngai and E. F. Westermann, "Statistical Multiplexing using MPEG-2 video encoders", IBM Thomas J. Watson Research Development, volume 43, No. 4, pp. 511-520, 1999.

[20] W.-C Feng, and J. Rexford, "A comparison of bandwidth smoothing techniques for the transmission of prerecorded compressed video" i*n Proc. of IEEE INFOCOM*, Kobe, Japan, pp. 58-66, 1997.

[21] J. Rexford, S. Sen, and A. Basso, " A Smoothing Proxy Service for Variable Bit-rate Streaming Video", i*n Proc. of IEEE Global Internet Symposium*, Rio de Janeiro, Brazil, vol.3, pp. 1823-1829, 1999.

[22] J.D. Salehi, Z.-L. Zhang, J. F. Kurose, and D. Towsley, "Supporting stored video: Reducing rate variability and end-to-end resource requirements through optimal smoothing", *IEEE/ACM Trans. Networking*, vol. 6, pp. 397-410, 1998.

[23] C.-W. Lin, J. Zhou, J. Youn, and M.-T. Sun, " MPEG Video Streaming with VCR Functionality", *IEEE Transaction on Circuits and System for Video Technology*, vol. 11, No. 3, pp. 415-425, March 2001.

[24] A. M. Tourapis, F. Wu, and S. Li, "Enabling VCR Functionalities in Streaming Media (Intelligent Interactive Streaming I^2 Stream)", Information Technology: Research and Education (ITRE), pp. 1-5, August 2003.

[25] K. Gao, Y. Zhang, W. Gao, and S. He, " Real-Time Scheduling Supporting VCR Functionality for Scalable Video Streaming", *in Proc. of the IEEE 14th Int. Symposium on Personal, Indoor and Mobile Radio Communication*, pp. 2711-2715, 2003.

[26] M. S. Chen and D. D. Kandlur, and P. S. Yu, "Support for Fully Interactive Playout in a Disk-Array-Based Video Server", *in Proc. of the 2nd ACM Int. Conf. on Multimedia, San Francisco*, CA, pp. 391-398, October 1994.

[27] M. Krunz and G. Apostolopoulos, "Efficient Support for Interactive Scanning Operations in MPEG-Based Video-On-Demand Systems", *ACM Multimedia Systems*, vol. 8, issue: 1, pp. 20-36, 2000.

[28] W. Feng, F. Jahanian, and S. Sechrest, "Providing VCR Functionality in a Constant Quality Video-on-Demand Transportation Service," *in Proc. of Int. Conf. On Multimedia Computing Systems (ICMCS)*, Hiroshima, Japan, pp. 127-135, June 1996.

[29] W. Liao and V. O. Li, " The Split and Merge Merge Protocol for Interactive Video-on-Demand," *IEEE Multimedia*, vol. 4, no. 6, pp. 51-62, 1997.

[30] Data-Over-Cable Service Interface Specification (DOCSIS), Radio Frequency Interface Specification, SP-RFIv1.1-I01-990211. [Online]. Available: http://www.cablemodem.com

[31] J. Farmer, W. Ciciora, and D. Large, *Modern Cable Television Technology: Video, Voice, and Data Communications*, San Francisco, CA: Morgan Kaufman, 1999.

[32] G. Abe, *Residential Broadband, Second edition*, Indianapolis, Cisco Press, 2000.

[33] P. Kyees, R. McConnell, and K. Sistanizadeh, ADSL: A New Twisted-Pair Access to the Information Highway, *IEEE Communications*, vol. 33, no. 4, pp. 52-59, April 1995.

[34] J. Cioffi, V. Oksman, J.-J. Werner, T. Pollet, P. Spruyt, J. Chow, and K. Jacobsen, Very-High-Speed Digital Subscriber Lines, *IEEE Communications Magazine*, vol. 37, no 4, pp. 72-79, April 1999.

[35] ITU SG-15 Recommendation G.983: High Speed Optical Access Systems Based on Passive Optical Network (PON) Techniques.

[36] R. E. Bellman, *Dynamic Programming*, Princeton, NJ: Princeton University Press, 1957.

[37] U.D. Black, *TCP/IP and Related Protocols*, New York, McGraw-Hill, 1995.

[38] D. D. Clark, "The Design Philosophy of the DARPA Internet Protocols," *in Proc. of ACM SIGCOMM '88 Conference*, pp. 106-114, 1988.

[39] P.T. Davis and C. R. McGuffin, *Wireless Local Area Networks*, New York, McGraw-Hill, 1995.

[40] E.W. Dijsktra, "A Note on Two Problems in Connection with Graphs," *Numerical Mathematics*, vol. 1. pp. 269-271, 1959.

[41] W. J. Goralski, *Introduction to ATM Networking*, New York McGraw-Hill, 1995.

[42] W. Fischer, E. Wallmeier, T. Worster, S.P. Davis, A. Hayter, "Data Communications Using ATM: Architectures, Protocols, and Resource Management," *IEEE Communication Magazine*, vol. 32, pp. 24-33, Aug 1994.

[43] S. Floyd and V. Jacobson, "Random Early Detection for Congestion Avoidance," *IEEE/ACM Transaction on Networking*, vol. 1, pp. 397-413 Aug 1993.

[44] C. Huitema, *Routing in the Internet*, Englewood Cliffs, NJ, Prentice Hall, 1995.

[45] P. Pancha and M. El Zarki, "MPEG Coding for Variable Bit Rate Video Transmission," *IEEE Communication Magazine*, vol. 32, pp. 54-66, May 1994.

[46] V. Paxson and S. Floyd, "Growth Trends in Wide-Area TCP Connections," *IEEE Network Magazine*, vol. 8, pp. 8-17, July/August 1994.

[47] A. S. Tanenbaum, *Computer Networks*, Upper Saddle River, NJ: Prentice Hall, 1996.

[48] L. Zhang, "RSVP: A New Resource ReSerVation Protocol," *IEEE Network Magazine*, vol. 7, pp. 8-18, 1993.

[49] R. Braden, Ed., L. Zhang, S. Berson, S. Herzog, S. Jamin, "Resource ReSerVation Protocol (RSVP)-Version 1 Functional Specification," IETF, RFC 2205, September 1997. [Online]. Available: http://www.faqs.org/rfcs/rfc2205.html.

[50] J. F. Kurose and K. W. Ross, *Computer networking: A top-down approach featuring the Internet*, New York, Addison-Wesley, 3rd edition, 2005.

[51] W. Stallings, *Data and Computer Communications, 6th edition*, Upper Saddle River, NJ: Prentice Hall, 2000.

[52] R. Steinmetz and K. Nahrstedt, *Multimedia: Computing, Communications, and Applications*, Upper Saddle River, NJ: Prentice Hall, 1995.

[53] J. Postel, "User Datagram Protocol (UDP)," IETF, RFC 768, August 1980. [Online]. Available: http://www.faqs.org/rfcs/rfc768.html.

[54] J. Postel, "Internet Protcol: DARPA Internet Program Protocol Specification," IETF, RFC 791, September 1981. [Online]. Available: http://www.faqs.org/rfcs/rfc791.html.

[55] J. Postel, "Transmission Control Protocol," IETF, RFC 793, September 1981. [Online]. Available: http://www.faqs.org/rfcs/rfc793.html.

[56] T. Socolofsky and C. Kale, "A TCP/IP Tutorial," IETF, RFC 1180, January 1991. [Online]. Available: http://www.faqs.org/rfcs/rfc1180.html.

[57] J. Moy, "OSPF Version 2," IETF, RFC 2328, April 1998. [Online]. Available: http://www.faqs.org/rfcs/rfc2328.html.

[58] Y. Rekhter and T. Li, "A Border Gateway Protocol 4 (BGP-4)," IETF, RFC 1771, March 1995. [Online]. Available: http://www.faqs.org/rfcs/rfc1771.html.

[59] L. Berger, "Generalized Multi-Protocol Label Switching (GMPLS) Signaling Resource ReserVation Protocol-Traffic Engineering (RSVP-TE) Extensions," IETF, RFC 3473, January 2003. [Online]. Available: http://www.faqs.org/rfcs/rfc3473.html.

[60] T. Berners-Lee, L. Masinter and M. McCahill, "URL: Uniform resource locators," IETF, RFC 1738, December 1994. [Online]. Available: http://www.faqs.org/rfcs/rfc1738.html.

[61] M. Handley and V. Jacobson, "SDP: Session Description Protocol," IETF, RFC 2327, April 1998. [Online]. Available: http://www.faqs.org/rfcs/rfc2327.html.

[62] K.A. Hua, Y. Cai, and S. Sheu, "Patching: A Multicast Technique For True Video-on-Demand Services," *in Proc. of 6th International Conf on Multimedia*, Sept 1998, pp.191-200.

[63] S.H.G. Chan and T.M. Ko, "Providing On-Demand Video Services Using Request Batching," *in Proc. of IEEE ICC' 98*, Atlanta, Georgia, pp. 1716-1722, June 1998.

[64] A. Bar-Noy, R. E. Ladner, "Competitive on-line stream merging algorithms for media-on-demand," *Journal of Algorithms*, v.48 n.1, p.59-90, August 2003

[65] E. Bommaiah, K. Guo, M. Hofmann, and S. Paul, "Design and implementation of a caching system for streaming media over the internet," in *Proc. of IEEE Real Time Technology and Applications Symposium*, pp. 111-121, Washington, DC, May 2000.

[66] Z.-N. Li and M. S. Drew, *Fundamentals of Multimedia*, ISBN: 0-13-061872-1, Pearson Education Inc., Pearson Prentice Hall, USA, 2004.

[67] Apple. [Online]. Available: http://www.apple.com/. (Accessed: August 19, 2005).

[68] RealNetworks. [Online]. Available: http://www.realnetworks.com/. (Accessed: August 19, 2005).

[69] Microssoft. [Online]. Available: http://www.microsoft.com/. (Accessed: August 19, 2005).

[70] D. Hoffman, G. Fernando, V. Goyal, M. Civanlar, "RTP Payload Format for MPEG1/MPEG2 Video," IETF, RFC 2250, January 1998. [Online]. Available: http://www.faqs.org/rfcs/rfc2250.html

[71] M. Orzessek and P. Sommer, *ATM & MPEG-2: Integrated Digital Video into Broadband Networks*, Upper Saddle River, NJ: Prentice Hall, 1998.

[72] U. Black, *Frame Relay Networks: Specifications and Implementations. 2nd ed.* New York: McGraw-Hill, 1996.

[73] ITU Telecommunication Standardization Sector (ITU-T). [Online]. Available: http://www.itu.int/ITU-T/. (Accessed: August 19, 2005).

[74] Moving Picture Experts Group (MPEG). [Online]. Available: http://www.chiariglione.org/mpeg/. (Accessed: August 19, 2005).

[75] M. Andrews and K. Munagala, "Online Algorithms for Caching Multimedia Streams," *in Proc. of 8th European Symposium on Algorithms*, pp. 64 –75, 2000.

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

[77] M. H. Kabir, E. G. Manning, G.. C. Shoja, "A CBR-streaming Scheme for VBR-encoded Videos," *IEEE PACRIM 05*, Victoria, Canada, pp. 213-216, August 2005.

[78] M. H. Kabir, E. G. Manning, G. C. Shoja, "Scalable Multimedia Streaming Model and Transmission Scheme for VBR-Encoded Videos," *in Proc. of 16th IASTED International Conference on Parallel And Distributed Computing And Systems (PDCS)*, MIT, Cambridge, MA, USA, pp. 867-872, 2004.

[79] M. Reisslein, F. Hartanto and K. W. Ross, "Interactive Video Streaming with Proxy Servers," *Proceedings of the first International Workshop on Intelligent Multimedia Computing and Networking (IMMCN),* Atlantic City, NJ, 2000.

[80] Jurassic Park I Video trace of MPEG-4 encoded. [Online]. Available: http://www-tkn.ee.tu-berlin.de/research/trace/ltvt.html. (Accessed: August 19, 2005).

[81] M.S. Chen and D.D. Kandlur, "Downloading and stream conversation: Supporting interactive playout of videos in a client station," *in Proc. of 2nd Int. IEEE Conf. Multimedia Computing and Systems*, pp. 73-80, 1995.

[82] S. Chen, "Reverse playback of MPEG video", U.S. Patent 5739862, April 14, 1998.

[83] S. J. Wee and B. Vasudev, "Compressed-domain reverse play of MPEG video streams," *in Proc. of SPIE Conf. Multimedia Systems and Applications*, pp. 237-248, Nov. 1998.

[84] S. J. Wee, "Reversing motion vector fields," *in Proc. of IEEE Int. Conf. Image Processing*, pp. 209-212, Oct. 1998.

[85] R. Kurceren and M. Karczewicz, "Improved SP-frame Encoding," document VCEG-M73, ITU-T Video Coding Expert Group Meeting, Autin, TX, 02-04, April 2001.

[86] R. Kurceren and M. Karczewicz, " New Macroblock Modes for SP-frames," document VCEG-047, ITU-T Video Coding Expert Group Meeting, Pattaya, Dec. 2001.

[87] X. Sun, F. Wu, S. Li, and R. Kurceren, " The improved JVT-B079 SP coding scheme," document JVT-C114, JVT Meeting, Fairfax, May 2002.

[88] J. K. Dey-Sircar, J. D. Salehi, J. F. Kurose, and D. Towsley, "Providing VCR Capabilities in Large-Scale Video Servers," *in Proc. of the 2nd ACM Int. Conf. On Multimedi*a, San Francisco, CA, pp. 25-32, October 1994.

[89] J. C. S. Lui and K.W. Law, " Load Balancing and VCR Functionalities Support via Subband Coding Techniques," *in Proc. of the 5th Int. Workshop on Network and Operating System Support for Digital Audio and Video*, Durham, NH, pp. 77-80, April 1995.

[90] P.J. Shenoy and H.M. Vin, "Efficient Support for Scan Operations in Video Servers," *in Proc. of 3rd Electronic ACM Multimedia Conf.*, pp. 131-140, Nov. 1995.

[91] P. P. White and J. Crowcroft, " Optimized Batch Patching with Classes of Service," *ACM SIGCOM Computer Communication Review*, pp. 21-28, October 2000.

[92] P. Salembier and J. R. Smith, "MPEG-7 Multimedia Description Schemes," *IEEE Transaction on Circuits and Systems for Video Technology*, vol. 11, no. 6, pp. 748-759, June 2001.

[93] H. M. Radha, M. van der Schaar, and Y. Chen, "The MPEG-4 Fine-Grained Scalable Video Coding Method for Multimedia Streaming over IP," *IEEE Transaction on Multimedia*, vol. 3, no. 1, pp. 53-68, March 2001.

[94] C.-Y. Lin, B. L. Tseng, M. Naphade, A. Natsev, and J. R. Smith, "MPEG-7 video automatic labeling system," *in the Proc. of the 11th ACM international conference on Multimedia.* Berkeley, CA, USA, pp. 98 – 99, November 2003.

[95] H.-W. Chen, J.-H. Kuo, W.-T. Chu, J.-L. Wu, "Action movies segmentation and summarization based on tempo analysis," *in the Proc. of the 6th ACM SIGMM international workshop on Multimedia information retrieval*, Nee York, NY, USA, pp. 251-258, October 2004.

[96] J. Nang, S. Hong, and Y. Ihm, "An efficient video segmentation scheme for MPEG video stream using macroblock information," *in the Proc of the 7th ACM international conference on Multimedia (Part 1*), Orlando, Florida, USA, pp. 23-26, October 1999.

[97] M. H. Kabir, E. G. Manning, and G. C. Shoja, "A Highly Scalable Multimedia Streaming Model and Transmission Scheme," Department of Computer Science, Victoria, BC, Canada, University of Victoria, IN Tech. Rep. DCS-288-IR, July 2004.

[98] M. H. Kabir, E. G. Manning, and G. C. Shoja, "An interactive scalable multimedia streaming scheme for VBR-encoded videos," *In the Proc. of the 3rd Annual Conference on Communication Networks and Services Research (CNSR2005)*, Halifax, Nova Scotia, Canada, pp. 145–150, May, 2005.

[99] M. H. Kabir, E. G. Manning, and G. C. Shoja, "An interactive and scalable streaming scheme for VBR-encoded videos," Department of Computer Science, University of Victoria, Victoria, BC, Canada, IN Tech. Rep. DCS-293-IR, July 2004.

[100] F. H. P. Fitzek and M. Reisslein, "MPEG-4 and H.263 video traces for network performance evaluation," TKN Technical Report-00-06, Telecommunication Network Group, Technical University Berlin, Berlin, October 2000.

[101] S. Acharya and B. Smith, "Middleman: A Video Caching Proxy Server," *in Proc. of the 10th International Workshop on Network and Operating System Support for Digital Audio and Video*, June 2000.

[102] M. Hoffman, T.S. Eugene Ng, K. Guo, S. Paul, H. Zhang, "Caching Techniques for Streaming Multimedia over the Internet," Technical Report, Bell Laboratories, BL011345-990409-04TM, April 1999.

[103] R. Stoica, D. Morris, Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup protocol for Internet applications," *IEEE/ACM Trans. on Networking*, vol. 11 no. 1, pp.17-32, February 2003.

[104] "Secure Hash Standard," U.S. Dept. Commerce/NIST, National Technical Information Service, Springfield, VA, FIPS 180-1, Apr. 1995.

[105] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: A distributed anonymous information storage and retrieval system," in *Proc. of the ICSI Workshop on Design Issues in Anonymity and Unobservability* (Berkeley, California, June 2000). http://freenet.sourceforge.net.

[106] Napster. [Online]. Available: http://www.napster.com/. (Accessed: August 19, 2005).

[107] Gnutella. [Online]. Available: http://gnutella.wego.com/. (Accessed: August 19, 2005).

[108] D. R. Karger, E. Lehman, F. Leighton, M. Levine, D. Lewin, and R. Panigrahy, "Consistent Hashing and Random Tree: Distributed Caching Protocols for relieving hot spots on the World Wide Web," *In Proc. of the 29th Annu. ACM Symp. Theory of Computing*, El Paso, TX, pp. 654-669, May 1997,

[109] D. Lewin, "Consistent Hashing and Random Tree: Algorithms for caching in Distributed Networks," Masters Thesis, Department of Electrical Eng. Comp Sci., Massachusetts Inst. Technology, Cambridge, 1998.

[110] M. H. Kabir, E. G. Manning, G. C. Shoja, "On-demand Segmentation and Buffer Provisioning Scheme," Department of Computer Science, University of Victoria, Victoria, BC, Canada, IN Tech. Rep. DCS-300-IR, May 2005.

[111] T. Berners-Lee, R. Fielding, L. Masinter, Uniform Resource Identifiers (URI): Generic Syntax, RFC-2396, August 1998.

[112] Y. Xiao and J. Rosdahl, "Throughput and Delay Limits of IEEE 802.11," *IEEE Communications Letters*, vol. 6, no. 8, pp. 355-357, August 2002.

[113] T. Doukoglou, D. Kagklis, S. Kontinis, S. Ioannou Skenter, P. Georgakopoulos, "Hi-Quality MPEG2 Video Delivery over Fixed Wireless Access Technologies (802.11b)," *in Proc. of 1st international workshop on streaming media distribution over the Internet (SMDI 04) of the Networking Conf.*, Athens, Greece, pp. 9-14 May 2004.

Appendix A

A.1 DOCSIS on Cable TV Networks

Cable TV networks are shared, wired networks built primarily for one-way television signal transmission. Multiple households connect to a common piece of cable. This is a natural topology for TV broadcast from the head-end to the households. It is not, however, a suitable topology for point-to-point Internet communications from the households. Cable networks require de-multiplexing of traffic from different households, provide bandwidth guarantees, and problem isolation. These problems have been solved, Data Over Cable Service Interface Specification (DOCSIS) [30][31][32] has been developed and cable networks are now used for two-way Internet communications using DOCSIS. Cable networks provide high-speed and always connected Internet access for computers with Ethernet or Universal Serial Bus (USB) ports through cable modems. A Cable TV network needs to set up a Cable Modem Terminal Server (CMTS) at its head-end to provide Internet access. Every household has to set up a cable modem to get Internet access from a cable TV network. CMTS implements Medium Access Control (MAC) protocol to control the channel sharing by the household cable modems. A CMTS connects itself to the public Internet through an IP router/switch at the head-end. Downstream traffic is modulated with QAM-64 or QAM-256 over 6MHz bandwidth. QAM-64 yields 30.3 Mbps and QAM-256 yields 42.9 Mbps. To make data service as compatible as possible with digital TV service, DOCSIS specifies MPEG-2 Transport Frame as the framing protocol, i.e., IP data packets are placed in the payload section of MPEG-2 frames. This allows data and video to be multiplexed on a common 6MHz channel. For upstream traffic, DOCSIS has five bandwidth options, such as 200kHz, 400kHz, 800kHz, 1.6MHz, and 3.2MHz, and two modulation options, such as QPSK and QAM-16. Upstream speed depends on what bandwidth and modulation options are used. The minimum speed is 320 kbps, when 200kHz bandwidth and QPSK modulation are used and the maximum speed is 10.24 Mbps, when 3.2MHz bandwidth and QAM-16

modulation options are used. The uses of cable downstream and upstream paths are shared among all the active modems connected to the same piece of cable. A 42.9 Mbps downstream channel and 10.24 Mbps upstream channel are shared among whomever communicating the head-end at any given moment. Therefore, a single individual user can experience less than the stated bit rate. Cable network's design promise is to provide full-service digital networking. In full-service, a consumer can receive all services (video, voice, and Internet) from a single carrier.

A.2 Digital Subscriber Line (DSL)

Digital Subscriber Line (DSL) [32][33][34] is a pair of wires that connects each subscriber to a local telephone company's central office (CO). DSL uses existing telephone lines to transmit data at speeds of over 100 or even 1000 times current dial-up modem service. A key difference between DOCSIS and DSL is that the users of DOCSIS in neighborhood share the copper cable, while DSL users do not. A single user gets the stated bit rate in DSL. Multiple flavors of DSL technologies are emerging. They are High data rate DSL (HDSL), Asymmetric DSL (ADSL), Single-Line DSL (SDSL), Integrated Services Digital Network (ISDN) DSL (IDSL), and Very high data rate DSL (VDSL). Collectively these are referred to as xDSL. New modulation, equalization, and error-control techniques make subscriber lines capable of transmission rates greater than 6 Mbps for distance up to 3 miles. xDSL speeds vary from 128 kbps (IDSL) to 51 Mbps (VDSL). xDSL distances from the central office to the subscribers vary from few hundred meters (300 to 1000) (VDSL) to few miles (IDSL). xDSL design promise is also to provide full-service (audio, video, data) to the consumers from a single carrier.

Telephone companies install DSL Access Multiplexers (DSLAMs) at their COs. A DSLAM houses multiple DSL Transmission-Unit-Central-Offices (TU-Cs). The TU-C is embedded in a line card in the DSLAM; there is one TU-C per subscriber. The DSLAM multiplexes traffic onto Fast Ethernet or ATM trunks. At the other end of the phone wire is a DSL Transmission-Unit-Remote (TU-R) at the subscriber site. The TU-R can support

a multi-drop or shared-home topology. In shared-home topology, only one TU-R is required even though multiple computers or televisions are connected through it within a home. The TU-C and TU-R engage in physical-layer negotiations between the home and the CO. A plain old telephone service (POTS) splitter is used at the subscriber site in front of TU-R, which enables the carriage of analog telephone and digital data/video service on the same wire. POTS splitter is a low-pass/high-pass filter that separates analog voice (the lowest 3.4 kHz) from DSL frequencies (start from 25 kHz).

A.3 Passive Optical Network (PON)

Passive Optical Network (PON) [35] is one kind of fiber to the home (FTTH) access network. It has a shared forward channel and point-to-point reverse channels with a MAC protocol to arbitrate return traffic contention. There are two service options in PON. One is the symmetric service of 155.52 Mbps (OC-3). The other one is asymmetric service of upstream 155.52 Mbps (OC-3) and downstream 622.08 Mbps (OC-12). The key elements in the PON are the Optical Line Terminator (OLT), Optical Distribution Network (ODN), and the Optical Network Unit (ONU). The OLT is like a DSLAM of a xDSL or CMTS of a cable network; it resides in a telephone company's CO. The ODN is the local loop distribution network, and the ONU is the in-home device that connects the home network to the distribution network (the ODN). The OLT couples the transport network, usually an ATM backbone, to the ODN and terminates and provides much of the control for the ODN. The OLT also enforces the MAC protocol for upstream bandwidth arbitration. The ODN is comprised of one or two single mode fibers and the passive optical components, primarily optical splitters. If a single mode fiber is used in ODN, bidirectional transmission is accomplished by use of wavelength division multiplexing (WDM) technique. The ODN offers one or more optical paths between one OLT and one or more ONUs. The ONU works like a cable modem of a cable network or like a TU-R of a DSL network. It provides the necessary functionality to connect the carrier fiber to the residence.

For downstream data, the OLT receives ATM [41] cells from the Internet through the Carrier ATM Transport Network and forwards them to the ODN. The splitter at the ODN takes the downstream optical wavelength and replicates it optically so that each ONU receives the same transmission. Therefore, a single laser transmitter at the central office can transmit downstream data to all residences. For upstream data, the ONU in the home receives data from the personal computer or from television set-top-box. The ONU requests permission of the OLT to transmit upstream. The OLT processes the request for bandwidth and grants credits to the ONU to transmit a specific number of ATM cells. Point-to-point upstream channels get credits for different time slots into which the clients can send return path data. All ONUs connected to common OLT use a common upstream frequency. Therefore, a single laser receiver at the central office can receive upstream data from all residences. The use of one laser transmitter and one laser receiver at the central office is an important cost-saver. This is why PON is seen as more practical than dedicated FTTH. Like Cable networks and DSL networks, PON's design goal is to provide full-services (audio, video, and data) to the consumers from a single carrier.

A.4 MPEG-2

MPEG-2 [12][13] achieves very high rates of compression by removing both spatial and temporal redundancies from video information. Spatial redundancy occurs because intensities of picture elements (called pels) are often replicated within a single video frame. Temporal redundancy arises when successive video frames display images of the same scene. MPEG-2 produces a compressed Intra-coded picture (I-frame) by removing the spatial redundancies present in a video frame without comparing it with other frames in the sequence. MPEG-2 produces a compressed Predictive-coded picture (P-frame) by removing both spatial and temporal redundancies present in a video frame. Temporal redundancies are predicted from a previous I- or P-frame. A Bidirectional-predicted picture (B-frame) is similar to a P-frame. Only difference is that the temporal redundancies are predicted from both a previous and a next I- or P-frames for B-frames.

Compressed frames are grouped into a group of picture (GOP). A GOP always starts with an I-frame. The typical frame sequence of a GOP with 12 frames is IBBPBBPBBPBB.

A.4.1 Intra Coding

MPEG standards use intra coding to create I-frames. Each picture divided into a number of blocks. Each block contains eight lines, each line holding eight samples of luminance or chrominance pixel values from a frame. Four blocks with luminance values, plus a number of blocks with chrominance values, form a macro-block. The number of chrominance blocks (2, 4, or 8) in macro-block depends on the sampling format used. MPEG applies discrete cosine transformation (DCT) on each block. DCT transforms luminance and chrominance data from the spatial domain to the frequency domain and separates high frequency information and low frequency information. The resultant DCT coefficient matrix represents information of the 8x8 block in frequency domain. DCT enhances compression by concentrating most of the energy in the signal in the lower spatial frequencies. The left upper corner of the matrix represents low frequency information while the right lower corner represents high frequency information. DCT transformation is not lossy. Inverse-DCT can restore original pixel values from DCT coefficients.

Quantization is used to achieve high compression by representing DCT coefficients with no greater precision than necessary by discarding information, which is not visually significant to human perception. Quantization is a lossy process. It divides each DCT coefficient by a constant. Quantization constants for an 8x8 block are defined in a quantization table.

After quantization, the 8x8 matrix of quantized coefficients is ordered into a one-dimensional array using the zig-zag sequence. This ordering facilitates entropy coding by placing low-frequency coefficients on the top of the vector. The DC coefficient, the top

left element in the matrix, is the measure of the average value of the 64 image samples of a block. Differential Pulse Code Modulation (DPCM) is applied on the DC coefficients of adjacent blocks. Run Length Encoding (RLE) is applied on the other coefficients, which usually have lots of zeros, of the matrix for further compression. Huffman coding may be used for further compression.

A.4.2 Inter Coding or Predictive Coding

MPEG standards use unidirectional predictive coding to generate P-frames. Each block in the current frame to be encoded is compared to areas of similar size from the preceding frame in order to find an area that is similar, i.e., the best matching block. The relative difference in locations of the matching blocks is called motion vector. The motion vector and the minor differences are coded instead of the block itself to achieve higher compression rate. This is called motion compensation.

For further and better compression, bidirectional predictive coding is used to generate B-frames. Areas just uncovered in the current frame are not predictable from the past frame, but they are easily predictable from the future frame. For this reason, bidirectional predictive coding searches in both past and future frames to find the best matching block and the motion compensations are computed from both directions.

A.4.3 MPEG-2 Bit Stream

The basic MPEG-2 stream is called Elementary Stream (ES). Each ES is an output of an MPEG-2 audio or video encoder and consists of only one type of stream, i.e., either video, or audio. A simple ES starts with a sequence header and contains many GOPs. Each GOP starts with a GOP header and contains a fixed or variable number of picture frames. Each picture frame has a picture-header and picture-data. A picture-data is divided into several slices to provide random access in a picture frame. Each slice contains an integral number of coded macro-blocks.

Video data is organized into access units, which represents a fundamental unit of encoding. For example, a complete encoded video frame can be an access unit. Each ES is given to an MPEG-2 processor to accumulate video data into a stream of Packetized Elementary Stream (PES). A PES packet can be a fixed or variable size block, with up to 64K bytes per block. A PES packet includes PES header and an integral number of ES access units. The fixed size PES packet header contains PES-start code, stream ID, PES-packet-length field, PES indicator, and Flags. PES indicators provide additional information, such as scrambling control, and priority, about the stream to assist the decoder. Flags define the presence of optional PES header fields. Optional PES header fields are: Presentation Time Stamp (PTS) and Decode Time Stamp (DTS), Elementary Stream Clock Reference (ESCR), Elementary Stream Rate, Trick Mode, Copyright Information, Cyclic Redundancy Code (CRC), and PES Extension Information.

Several PES streams can be multiplexed into an MPEG system stream. MPEG-2 allows two forms of multiplexing: MPEG Program Stream and MPEG Transport Stream. MPEG Program Stream is group of tightly coupled PES packets referenced to the same time base. This stream is suitable for transmission in a relatively error-free environment and enables easy software processing of the received data. In MPEG Transport Stream, each PES packet is broken into fixed-sized (188 bytes) transport packets forming a general-purpose way of combining one or more streams, possibly with independent time base. Transport stream is suitable for transmission in an error-prune and lossy environment.

A.5 RTSP

Real time Streaming Protocol (RTSP), RFC-2326 [2], is used to establish and control a streaming session. RTSP is an application-level request-response protocol, which is intentionally similar in syntax and operation to Hypertext Transfer Protocol version 1.1

(HTTP/1.1) [8], which is used by web browsers to get data from web servers. However, RTSP differs in a number of important aspects from HTTP:

- RTSP has the notion of session built into the protocol. HTTP has no notion of session.
- A RTSP server needs to maintain state by default in almost all cases, as opposed to the stateless nature of HTTP.
- Both RTSP server and client issue requests. Only HTTP client issues requests and HTTP server always responds to client requests.
- In RTSP data transmission is done out-of-band by a different protocol. HTTP request and response messages carry the data if there is any.

RFC-2326 recommends the use of RTSP over a reliable connection based transport level protocol such as TCP. We describe presentation description, RTSP URL, RTSP request-response message formats, and RTSP request methods in the following sections.

A.5.1 Presentation Description

Session Description Protocol (SDP) (RFC 2327) [61] or other different formats can be used to create a presentation description. A complex presentation description may contain one or more presentations, each of which maintains a common time axis. A simple presentation description contains a description of the media streams making up the presentation, including their encodings, language, and other parameters that enable the client to choose the most appropriate combination of media. The description also enumerates the transport protocols, e.g., RTP, that the server is capable of supporting. A RTSP URL identifies each media stream, which is individually controllable by RTSP, in a presentation.

A.5.2 RTSP URL

The "rtsp", "rtsps", and "rtspu" schemes are used to refer to network resources via RTSP protocol. The RTSP URL is case sensitive. Following is the syntax for RTSP URLs.

rtsp_URL = ("rtsp:"| "rtspu:"| "rtsps:") "//" host [":" port][abs_path ["?" query]]

RTSP commands must be issued via a reliable protocol, e.g., TCP under "rtsp" scheme while they can be issued via an unreliable protocol, e.g., UDP under "rtspu" scheme. The scheme "rtsps" assumes a reliable and secured transport protocol. If the port is not given, port 554 is assumed by default. For example, the RTSP URL: "rtsp://media.domain.com/twister/audio_track", identifies the audio stream within the presentation "twister", which can be controlled via RTSP requests issued over a TCP connection to port 554 of host "media.domain.com".

A.5.3 RTSP Request and Response Message Formats

RTSP is a text based request-response protocol. RTSP request or response is issued as a RTSP request or response message. A RTSP request message starts with a Request-Line. Request-Line contains the name of a RTSP method, the object the method is operating upon, and the RTSP version number. After the Request-Line a RTSP request message may have any or all of the following headers: General-header, Request-header, and Entity-header. The request message header ends with a carriage return and a line feed character. If there is any message body it immediately follows the header. General-header may contain any of the following header fields; Cache-Control, Connection, CSeq, Date, Timestamp, and Via. Request-header may contain any of the following header fields: Accept, Accept-Encoding, Accept-Language, Authorization, Bandwidth, Blocksize, From, If-Modified-Since, Proxy-Require, Range, Referrer, Require, Scale, Session, Speed, Supported, Transport, and User-agent. Entity-header may contain any of the following header fields: Allow, Content-Base, Content-Encoding, Content-Language, Content-Length, Content-Location, Content-Type, Expires, Last-Modified, and extension-header.

A RTSP response message starts with a Status-Line. A Status-Line contains RTSP version number, a Status-Code, and a Reason-Phrase. Like request message, a response message may have General-header, Response-header, Entity-header, and message body after the Status-Line. Response-header may contain any of the following header fields: Accept-Ranges, Location, Proxy-Authenticate, Public, Range, Retry-After, RTP-Info, Scale, Session, Server, Speed, Transport, Unsupported, Vary, and WWW-Authenticate.

The Status-Code in the Status-Line is a 3-digit integer result code against a request to a RTSP server. The Reason-Phrase is a short textual description of the Status-Code. The first digit of the Status-Code defines the class of response. There are five classes of responses as follows:

1XX: Informational-Request received, continuing process

2XX: Success-The action was successfully taken

3XX: Redirection-Further action must be taken in order to complete the request

4XX: Client Error-The request contains bad syntax or cannot be fulfilled

5XX: Server Error- The server failed to fulfill an apparently valid request

A.5.4 RTSP Methods

Every RTSP request includes a RTSP method in its Request-Line. A RTSP method indicates the operation to be performed on the resource identified by the RTSP URL. RTSP methods are case-sensitive. We have already mentioned SETUP, PLAY, and TEARDOWN methods in Chapter 2. The other methods in RTSP are DESCRIBE, GET_PARAMETER, OPTIONS, PAUSE, PING, REDIRECT, and SET_PARAMETER. Here, we briefly describe all RTSP methods.

OPTIONS: The OPTIONS method is used to request for information about the communication options available on the request/response chain identified by the Request-URL. The Public header field must be included in the response to indicate the list of methods that are supported by the server.

DESCRIBE: The DESCRIBE method is used to request the description of a presentation or media object identified by the request URL from a server. It can use the Accept header field to specify the description formats that the client understands. The server responds with a description of the requested resource in the body of the response message. OPTIONS and DESCRIBE methods can be used in the initialization phase before starting a RTSP session.

SETUP: The SETUP method is used to create a new RTSP session or to add a media stream to a previously established session. When the SETUP method is used to create a new session the server generates a session identifier in response to a successful SETUP request and sends the session identifier back to the client using the Session header field in its response message. All other successive client requests, such as PLAY, PAUSE, TEARDOWN, GET_PARAMETER, SET_PARAMETER, REDIRECT, and PING, pertaining to this session must carry this identifier using the Session header field. In order to add a media to a previously established session the SETUP request from a client enumerates the session's identifier using Session header field. In a SETUP response the server includes the Accept-Ranges header to indicate the time formats that are acceptable to use for this media resource. The SETUP request for a RTSP URL specifies the transport mechanism to be used for the streamed media. The Transport header field in the SETUP request specifies the transport parameters acceptable to the client for data transmission; the Transport header field in the response contains the transport parameters selected by the server. A client can use the SETUP method to change the transport parameters of already set up media stream.

PLAY: The PLAY method is used to request the server to start sending data via the transport mechanism specified in SETUP. A PLAY request without a Range header field will start playing a stream from the beginning of the stream. If a stream has been paused via PAUSE request, a PLAY request without a Range header field will resume media playback from the paused point. A PLAY request with a Range header field will start playback from the beginning of the specified range. Multiple ranges can be specified in one PLAY request. The Scale header field can be used to specify playback mode. The absence of the Scale header field or a scale value 1 is used to specify normal playback mode. A scale value more than 1, e.g., 2.0, can be used to specify fast forward playback mode and a scale value less than 1, e.g., 0.5, can be used to specify slow motion.

PAUSE: The PAUSE method is used to interrupt the stream delivery or playback temporarily. The PAUSE request can use a Range header field to specify a pause point. In this case, stream delivery will be interrupted when it reaches the pause point. If the Range header field is not used, stream delivery is interrupted immediately on receipt of the message and the pause point is set to the current normal playtime. A PLAY request, if issued before server time out, resumes playback from the pause point.

TEARDOWN: The TEARDOWN method is used either to halt a RTSP session or to remove a media stream from a RTSP session. Aggregated (presentation) control URL is used to halt a session. A successful request results in that media delivery is immediately halted, the resources have been freed, and the session state is destroyed. Media control URL is used to free a media stream from a session. In this case, the delivery of the freed stream is immediately halted and the session continues if there is at least one stream still left in the session. If no stream is left in the session, the session is also halted.

GET_PARAMETER: This method retrieves the value of the parameters, such as "packet_received" and "jitter", of a presentation or stream specified in the URL.

SET_PARAMETER: This method sets the value of a parameter for a presentation or stream specified by the URL.

REDIRECT: The server informs the client that it must connect to another server location using this method. The Location header field specifies the URL to where the client should connect.

PING: This method provides both the server and the client with a bi-directional mechanism to check each other's liveliness.

A.5.5 RTSP Session States

There are three possible states for a RTSP session: Init, Ready, and Play. Init state means either the session is not established yet or there is no media stream left in the session that can be played. A successful SETUP request-response method can change a session's state from Init to Ready. Ready state means the session is established and there is at least one media stream in the session ready to be played. A successful PLAY request-response method can change a session's state from Ready to Play. Play state means the client is now playing the media streams of the session.

A.6 RTP

Real-time Transport Protocol (RTP) and RTP Control Protocol (RTCP), RFC 1889 [1], are used to transfer streaming data from the source to the destination. We describe RTP packets, MPEG stream encapsulation into RTP packets, and RTCP in the following sections.

A.6.1 RTP Packets

A RTP sender encapsulates a media chunk or real-time data within an RTP packet and sends this packet to the receivers. The receiver extracts the media chunk from the RTP packet and passes the chunk to the media player for decoding and rendering. An RTP packet contains an RTP header and a media chunk. Each chunk of real-time data, receipt at the receiver, needs to carry timing information, such as the sampling instant, in order to regenerate correct playback sequence. If a large media chunk, sampled at an instance, does not fit into a single RTP packet due to the limit imposed by the maximum transmission unit (MTU) of a network, several RTP packets are used to transmit it over the constraint network. For example a video frame might need to be broken into several RTP packets. For this reason, sequence number is necessary to number the packets since some packets may have the same sampling timestamp. An RTP packet carries the timestamp and the sequence number along with other important information in its header. Other important information includes payload type and source identification of the stream. There are some other fields in an RTP header, such as version, padding, extension, contributing source count, marker, and contributing source list.

The version field identifies the version of RTP. If padding is set, the packet contains one or more additional padding octets at the end, which are not part of the payload. The last octet of the padding contains a count of how many padding octets are there. The extension field indicates header extension. The marker field can be used to indicate frame boundaries. The contributing source count contains the number of contributing sources in contributing source list. Contributing source list is left empty and the contributing source count is set zero for a regular source.

Payload-type indicates the type of encoding used for the stream. For example payload-type number 0 is used for PCM μ-law audio, 14 is used for MPEG audio, 32 is used for MPEG-1 video, and 33 is used for MPEG-2 video.

Source identification is called Synchronization Source Identifier (SSRC) and is used to identify the source of the RTP stream. If a multimedia transmission from a source involves both audio and video streams and the encoder does not multiplex these streams, i.e., audio and video streams remain separate; they are transmitted in separate RTP streams. Each RTP stream in the transmission will have distinct SSRC. SSRC is a 32-bit number chosen randomly when a new stream is started. The probability that two streams get assigned the same SSRC is very small. If it happens, two sources pick two new numbers. An RTP receiver also needs to pick a SSRC to send its receiver report to the sender using RTCP protocol. The SSRC for all the end systems in an RTP session needs to be distinct.

The timestamp reflects the sampling instant of the first byte in the RTP data packet. The receiver uses timestamp in order to remove packet jitter introduced in the network and to provide synchronous playback. The timestamp is generated from a sampling clock at the sender. The timestamp clock continues to increase at a constant rate even if the source is inactive.

The sequence number is incremented by one for each RTP packet sent. RTP guarantees neither packet delivery nor ordered delivery. The receiver uses sequence number to detect packet loss and to restore packet sequence. An RTP receiver neither sends an acknowledgement when a packet is received nor asks for a packet retransmission when a packet is lost. An RTP receiver may conceal the lost data by using a non-RTP mechanism.

A.6.2 MPEG Stream Encapsulation into RTP Packets

RFC 2250 [70] has described how to encapsulate MPEG video streams into RTP packets. Payload-type, marker, and timestamp fields of the RTP header are used in order to encapsulate an MPEG stream. The payload-type indicates whether it is MPEG-1 System

Stream, or MPEG-2 Program Stream, or MPEG-2 Transport Stream. The marker is set 1 whenever the timestamp is discontinuous, i.e., when a sender switches from one data source to another. The timestamp represents the target transmission time for the first byte of the packet. MPEG stream's Program Clock Reference (PCR) or System Clock Reference is used as the RTP timestamp. The RTP timestamp is not passed to the MPEG decoder. It is mainly used to estimate and reduce any network-induced jitter and to synchronize relative time drift between the transmitter and receiver. When MPEG-2 Transport Streams are used an RTP payload contains an integral number of MPEG transport packets.

A.6.3 RTCP

In an RTP communication, participants periodically send RTCP (RFC 1889) [1] packets to convey feedback on quality of data delivery and information about the participants. There are five types of RTCP packets: receiver report (RR), sender report (SR), source description items (SDES), BYE, and application specific functions (APP). By sending a RR packet, a receiver reports reception quality feedback about the data delivery, including the highest packet number received, the number of packets lost, the fraction of packet transmission lost since last RR, inter-arrival jitter, and timestamp to compute the round-trip delay between the sender and the receiver. Senders can adjust their transmission based on the receiver report feedback. A sender reports inter-media synchronization information, number of packets sent, and the number octets sent by sending an SR packet. The inter-media synchronization information is an indication of real time and the corresponding RTP timestamp. A sender sends SDES packets to describe itself. SDES packets contain textual information called canonical names (CNAMEs) as globally unique identifiers of the participants. CNAME provides the binding from the new SSRC identifier to an identifier that remain constant for a source when a SSRC identifier of the source changes due to conflict in SSRC space or due to program restart. It also provides a binding across multiple media tools used by one participant in a set of related RTP communications. By sending a BYE packet a participant indicates end of its participation. APP packets are intended for experimental

use as new application and new features are developed. RCTP packets are stackable. A stacked RTCP packet may contain several types of RTCP packets; the first RTCP packet must always be either SR or RR; an SDES packet containing a CNAME must be included; BYE should be the last packet. RTCP prevents the control traffic from overwhelming network resources by limiting the control traffic to at most 5% of the overall RTP traffic. It adjusts the RTCP packet-generating rate based on the number of participants.

A.7 Chord Placement and Lookup Protocols in a Peer-to-Peer System

Chord [103] is a suite of scalable protocols for placement and lookup of data in a dynamic peer-to-peer system with frequent node arrivals and departures. Given a key, it maps the key onto a node. The node is responsible for storing the data associated with the key. It uses consistent hashing [108][109] to assign keys to the nodes. Consistent hashing balances load through mapping roughly the same number of keys onto each node. It also requires little movement of keys when nodes join and leave the system. Chord nodes also keep routing information about other nodes and the routing table is distributed. A Chord node may need to communicate with other nodes in order to perform lookup. Chord protocols are simple and perform well.

A.7.1 Placing keys onto Chord nodes using Consistent Hashing

The consistent hash function assigns each node and each key an h-bit *identifier* using SHA-1 [104] as a base hash function. By hashing the node's IP address a node identifier is assigned. Similarly, by hashing the key a key identifier is assigned. The identifier length is chosen large enough to make the probability of two nodes or keys hashing to the same identifier negligible. Identifiers are ordered on an *identifier circle* modulo 2^h. Key k is assigned to the first node whose identifier is equal to or follows (the identifier of) k in the identifier space. This node is called the *successor node* of key k, denoted by $successor(k)$. If identifiers are represented as a circle of numbers from 0 to $2^h - 1$, then

successor(k) is the first node clockwise from k. The identifier circle is called the *Chord ring*. Stocia et al. [103] shows a Chord ring with $h = 6$ as shown in Figure A.1. This Chord ring has ten nodes and stores five keys. The successor of identifier 10 is node 14, so key 10 would be located at node 14. Similarly, keys 24 and 30 would be located at node 32, key 38 at node 38, and key 54 at node 56.

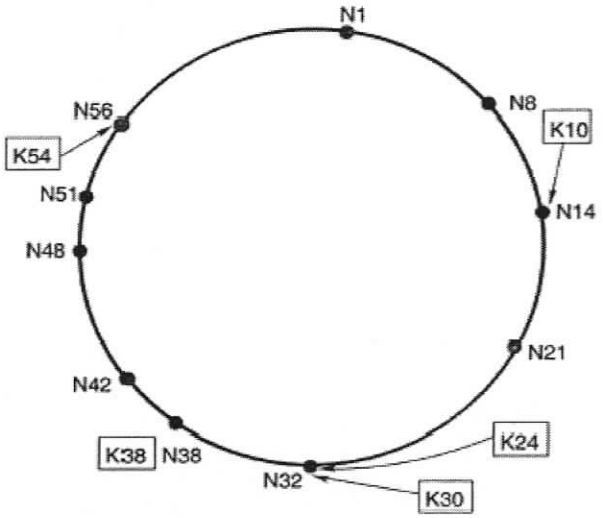

Figure A.1 Chord Ring consisting of ten nodes and storing five keys

Chord is designed to let nodes enter and leave the network with minimal disruption. To maintain the consistent hashing mapping when a node N joins the network, certain keys previously assigned to N's successor now become assigned to N. When node N leaves the network, all of its assigned keys are reassigned to N's successor. No other changes in the assignment of keys to nodes need to occur. In Figure A.1, if a new node were to join with identifier 26, it would capture the key with identifier 24 from the node with identifier 32. If node 38 leaves, the key with identifier 38 will be assigned to node 42.

A.7.2 Key Lookup in the Chord Nodes

In order to accelerate lookups, Chord maintains routing information at each node. Each node maintains a routing table with up to h entries called the *finger table*. The i^{th} entry in the table at node N contains the identity of the *first* node s that succeeds by at least 2^{i-1} on the Chord ring, i.e., $s = successor(N + 2^{i-1})$, where $1 \le i \le h$. Node s is called the i^{th} *finger* of node N. A finger table entry includes both the Chord identifier and the IP address (and port number) of the relevant node. The first finger of a node N is the immediate successor of N on the ring and can be referred as N 's *successor*. Stocia et al. [103] shows the finger table of node 8 of Figure A.1. Here, we show that finger table in Figure A.2. The first finger of node 8 points to node 14, as node 14 is the finger of node 8 that succeeds $(8 + 2^0) \bmod 2^6 = 9$. In Figure A.2 node 14 is node 8's successor. Similarly, the last finger of node 8 points to node 42, as node 42 is the first node that succeeds $(8 + 2^5) \bmod 2^6 = 40$.

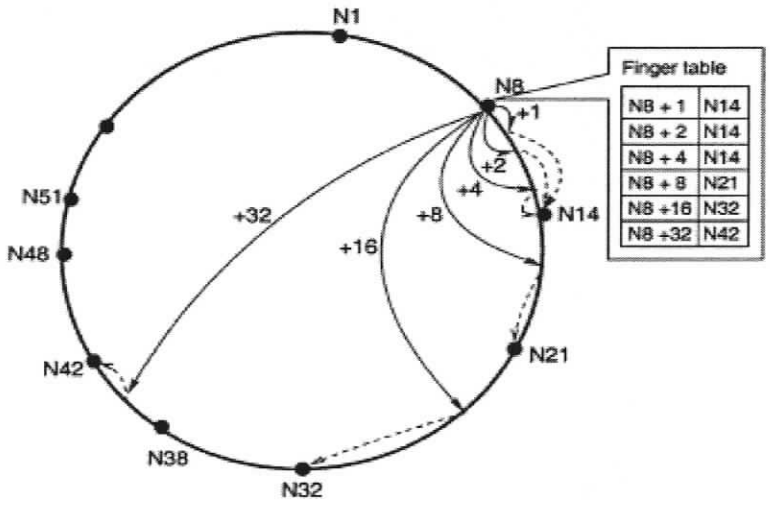

Figure A.2: Finger table entries for node 8

To lookup a key in another node from a node N Chord searches the finger table at N. If the *id* of the key falls between N and its successor, the search finishes and returns N 's

successor. Otherwise, Chord searches N's finger table for the node N', whose ID most immediately precedes *id*, and then starts searching the finger table at N'. The reason behind this choice of N' is that the closer N' is to *id*, the more it will know about the identifier ring in the region of *id*. As an example, consider the Chord ring in Figure A.3 (also taken from [103]), and suppose node 8 wants to find the successor of key 54. Since the largest finger of node 8 that precedes 54 is node 42, node 8 will ask node 42 to resolve the query. In turn, node 42 will determine the largest finger in its finger table that precedes 54, i.e., node 51. Finally, node 51 will discover that its own successor, node 56, succeeds key 54, and thus, will return node 56 to node 8. Since each node has finger entries at power of two intervals around the identifier circle, each node can forward a query at least halfway along the remaining distance between the node and the target identifier.

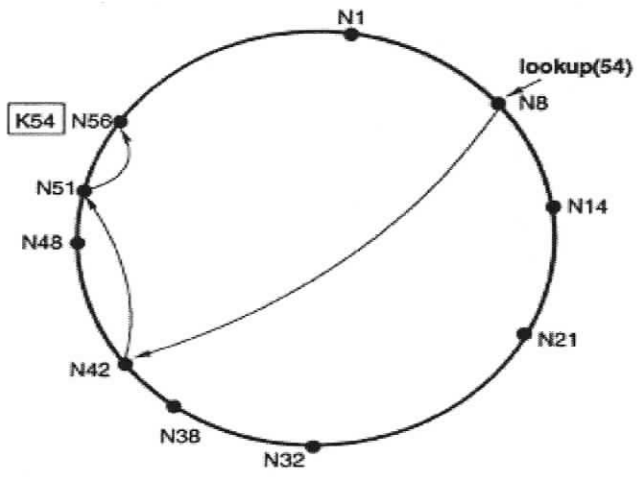

Figure A.3: Path of a query for key 54 starting at node 8

A.7.3 Handling Chord node Join, Failure and Departure

In order to ensure that lookups execute correctly as the set of participating nodes changes, Chord ensures that each node's successor pointer and finger table are up to date. Chord uses a "*stabilization*" protocol that each node runs periodically in the background and which updates Chord's finger tables and successor pointers. Details of the stabilization protocol can be found in [103].

Vita

Surname:	Kabir	Given Names:	Md Humayun
Place of Birth:	Dhaka, Bangladesh		

Educational Institutions Attended:

Bangladesh University of Engineering and Technology (BUET)	1988 - 1993
Bangladesh University of Engineering and Technology (BUET)	1996 - 1998
University of Victoria	2002 – 2005

Degrees Awarded:

M.Sc. in Computer	Science and Engg	BUET	1998
B.Sc. in Computer	Science and Engg	BUET	1993

Honours and Awards:

NewMIC Research Scholarship	2002 - 2003

Publications

1. M. H. Kabir, E. G. Manning, G. C. Shoja, "On-demand Segmentation and Buffer Provisioning Scheme for Scalable and Interactive Video Streaming Scheme", IEEE INFOCOM 06, Barcelona, Spain, 2006 (submitted).
2. M. H. Kabir, E. G. Manning, G. C. Shoja, "A Scalable Multimedia Streaming Scheme with CBR-transmission of VBR-encoded Videos over the Internet", International

Journal of Interactive Technology and Smart Education (ITSE), Special Issue: Streaming Content Distribution Networks for e-Learning and e-Entertainment, Troubador Publishing Ltd., Leicester, UK (accepted, to be published in 2006).

3. M. H. Kabir, E. G. Manning, G. C. Shoja, "Collaborative Proxies for Interactive Scalable Multimedia Streaming Scheme for VBR-encoded Videos," Department of Computer Science, University of Victoria, Victoria, BC, Canada, IN Tech. Rep. DCS-302-IR, August 2005.

4. M. H. Kabir, E. G. Manning, G. C. Shoja, "On-demand Segmentation and Buffer Provisioning Scheme," Department of Computer Science, University of Victoria, Victoria, BC, Canada, IN Tech. Rep. DCS-300-IR, May 2005.

5. M. H. Kabir, E. G. Manning, G.. C. Shoja, "A CBR-streaming Scheme for VBR-encoded Videos," *IEEE PACRIM 05*, Victoria, Canada, 2005.

6. M. H. Kabir, Eric G. Manning, and Gholamali C. Shoja, "An Interactive Scalable Multimedia Streaming Scheme for VBR-encoded Videos", In the Proceedings of Third Annual Conference on Communication Networks and Services Research (CNSR2005), Halifax, Nova Scotia, Canada May, 2005.

7. M. H. Kabir, Eric G. Manning, and Gholamali C. Shoja, "An Interactive and Scalable Streaming Scheme for VBR-encoded videos", Technical Report: DCS-293-IR, Department of Computer Science, University of Victoria, Victoria, BC, Canada, July 2004.

8. M. H. Kabir, Eric G. Manning, and Gholamali C. Shoja, "A Highly Scalable Multimedia Streaming Model and Transmission Scheme", Technical Report: DCS-288-IR, Department of Computer Science, University of Victoria, Victoria, BC, Canada, July 2004.

9. M. H. Kabir, Eric G. Manning, and Gholamali C. Shoja, "Scalable Multimedia Streaming Model and Transmission Scheme for VBR-Encoded Videos" in the Proc. of 16th IASTED Int. Conf. on Parallel And Distributed Computing And Systems (PDCS), MIT, Cambridge, MA, USA, Nov. 2004.

10. Md Humayun Kabir, Eric G Manning, and Gholamali C Shoja, "Request-routing Trends and Techniques in Content Distribution Networks", In the Proceedings of ICCIT 02, Dhaka, Bangladesh, December 2002.

11. Md.Humayun Kabir, and Chowdhury Mofizur Rahman, "A new Design Methodology for getting Normalized Relations in Relational Databases", In the Proceedings of ICCIT 98, Dhaka, Bangladesh, December 1998.

12. Md.Humayun Kabir, and Chowdhury Mofizur Rahman, "Database Normalization using Machine Learning Techniques", In the Proceedings of NCCIS 97, Dhaka, Bangladesh, December 1997.

13. Maria Wahid Chowdhury, Chowdhury Mofizur Rahman, and Md. Humayun Kabir, "Distributed Index: Algorithms and Models", In the Proceedings of ICCIT 2000, Dhaka, Bangladesh, December 2000.

14. Mohammad Rashedur Rahman, Sajib Barua, and Md. Humayun Kabir, " A Study on Neural Network Technique for Medical Diagnostic Reasoning", In the Proceedings of ICCIT 2000, Dhaka, Bangladesh, December 2000.

15. Md. Humayun Kabir, "Database Characterization using Induction", M. Sc. Engineering Thesis, Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh, 1998.